Towards Bidirectional Higher-Order Transformation for Model-Driven Co-evolution

Bernhard Hoisl[1,2(✉)], Zhenjiang Hu[3], and Soichiro Hidaka[3]

[1] Institute for Information Systems and New Media, Vienna University of Economics and Business (WU Vienna), Vienna, Austria
bernhard.hoisl@wu.ac.at
[2] Secure Business Austria Research (SBA Research), Vienna, Austria
[3] National Institute of Informatics (NII), Tokyo, Japan
{hu,hidaka}@nii.ac.jp

Abstract. In model-driven development (MDD), numerous metamodels, models, and model transformations need to be taken into account. These MDD-based artifacts—although highly interdependent—are autonomously maintained. Changes in one artifact (e.g., in a model) are not automatically reflected in other dependent artifacts (e.g., in a model transformation). The barrier for a tight integration of MDD-based artifacts stems from two limitations of current approaches. On the one hand, model transformations are unidirectional and changes can be propagated in one direction only. On the other hand, changes can only be propagated into output artifacts of transformations, not into transformation definitions themselves. In order to overcome these co-evolution problems, our approach is based on establishing bidirectional transformations (BX) between modeling artifacts and on applying higher-order transformations (HOTs) on first-class model representations of transformation specifications. In this paper, we present a generic approach and provide initial prototypes for an integrated tool support which integrates BX into well-established Eclipse-based MDD frameworks, thereby neither being restricted to a specific modeling nor model transformation language.

Keywords: Model-driven development · Model co-evolution · Bidirectional transformation · Higher-order transformation

1 Introduction

In model-driven development (MDD; see, e.g., [2,3]), numerous models and transformations on different abstraction levels need to be taken into account. The high number of models involved originate from a layered modeling architecture (i.e. metamodels, MMs) as well as from refinements (i.e. transformations) from generic to implementation-centric model representations [4]. On the one hand,

This is an extended version of the paper published as Hoisl et al. [1].

S. Hammoudi et al. (Eds.): MODELSWARD 2014, CCIS 506, pp. 153–167, 2015.
DOI: 10.1007/978-3-319-25156-1_10

the need for model transformations is inherent to the abstraction mechanism in MDD to represent platform-specific concepts (e.g., statements in a programming language) as platform-independent models [5]. On the other hand, model transformation necessities stem also from, for instance, changes in MMs (e.g., changes from the original MM to a new MM are implemented by model-to-model transformations, M2M) or the support for multiple platforms (e.g., platform-specific textual representations, such as, source code or configuration and deployment documents, are provided by different model-to-text transformations, M2T).

MDD-based artifacts are frequently subject to change and evolve over time [6]. In most cases, the evolution of (meta)models and model transformations is a manual process [7]. Individually maintain and manually evolve MDD specifications is a tedious and error-prone task [8,9]. For instance, consider an evolution of a MM and accompanying constraints. First, all instance models need to be migrated in order to conform to the new MM definition. Furthermore, all model transformations need to be adapted (e.g., due to model type changes). Moreover, tests need to be rewritten to check that the generated source code fulfills the specified constraints.

The artifacts which make up a MDD process (models, M2M/M2T transformations, model and transformation constraints etc.)—although highly interdependent—are autonomously maintained. Changes in one artifact (e.g., in a model) are not automatically reflected in other dependent artifacts (e.g., in a M2T transformation). The barrier for a tight integration of MDD-based artifacts stems from two limitations of current approaches: (1) Model transformations are unidirectional and changes can be propagated in one direction only (e.g., a model change is reflected in generated code); (2) changes can only be propagated into output artifacts of transformations (e.g., models), not into transformation definitions themselves.

In order to overcome these co-evolution problems, our approach, on the one hand, is based on (1) establishing bidirectional transformations (BX) between modeling artifacts (see, e.g., [8]). *BX* is a mechanism for maintaining the consistency of two (or more) related sources of information. A BX between two sources of information A and B (e.g., two different models) comprises a pair of unidirectional transformations: one from A to B (*forward transformation*) and another from B to A (*backward transformation*) [10].

On the other hand, we apply (2) higher-order transformations (HOTs) on first-class model representations of transformation specifications [11]. A *HOT* "is a model transformation such that its input and/or output models are themselves transformation models. [...] This demands the representation of the transformation as a model conforming to a transformation MM" [11].

In this way, we are able to propagate changes in two directions (1): From a source model to a target model and vice versa. These changes can be propagated into models on the same or on different abstraction levels. Furthermore, we ensure not only the co-evolution of models, but (2) model transformations, as well. We represent transformation definitions as models and are able to propagate changes into horizontal and vertical model transformations (i.e. transformations between models on the same and on different abstraction levels).

Our contributions are as follows:

- *A Method for MDD-based Co-evolution:* Our approach of bidirectional higher-order transformation (B-HOT) for the co-evolution of model artifacts builds on former work [12–15]. This paper presents first enhancement steps which will allow for coupling, synchronization, and tracing of all model artifacts involved in a MDD process.
- *Integrated Tool Support:* We provide initial prototypes for an integrated MDD-based tool support for B-HOTs via the Eclipse IDE. Our implementations build on well-established MDD tools (e.g., Eclipse EMF [16], ATL [17], Epsilon [18])[1].
- *Conformance between BX and MDD:* To facilitate reproduction and transferability, we present an approach independent of any transformation language and we prepare for generalizations according to OMG specifications, for example, MOF [19], QVT [20], MOFM2T [21]. Besides integrated BX and MDD tooling, we also want to contribute to establish a common terminology to bridge the gap between the BX and MDD communities [10,22].

The remainder of this paper is structured as follows. Section 2 reviews traditional model-driven architectures and explains why current approaches cannot sufficiently cope with the co-evolution of multiple MDD-based artifacts. Section 3 describes our approach to overcome the shortcomings of current methods. Requirements of our approach are discussed in Sect. 4. Our initial MDD-based developments are briefly explained in Sect. 5. Section 6 concludes the paper by discussing implications and mentioning limitations of our approach as well as pointing to ongoing and future work.

2 Current Approaches

A traditional model-driven architecture (MDA), as proposed by the OMG [23,24] and as supported by a variety of tools, is sketched in Fig. 1. MMs provide the reference frame to which instance models must conform to, for example, a UML class model conforms to its MM defined in the UML specification [25]. M2M transformations are applied over one or more input models with the purpose of generating one or more output models conforming to the same or different MMs. A typical M2M transformation example is the generation of platform-specific models (PSMs) from platform-independent models (PIMs). As models are a means for abstraction, they mostly do not capture enough implementation details to be directly executable. Hence, M2T transformations generate textual artifacts (e.g., source code, configuration documents) which can be deployed on a specific platform.

When a MDD-based artifact evolves, changes must be reflected in all dependent (meta)models, transformations, and platform artifacts. The complexity of

[1] All software artifacts are publicly available at http://www.biglab.org and http://nm.wu.ac.at/modsec.

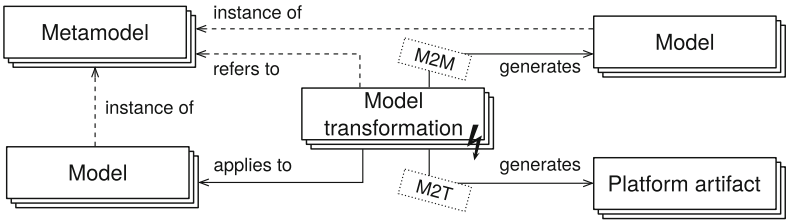

Fig. 1. Traditional model-driven architecture.

change operations increases with the number of different MMs and models (e.g., due to different modeling languages), M2M transformations (e.g., due to intermediary model representations), and M2T transformations (e.g., due to multiple supported platforms) involved. Current approaches cannot sufficiently cope with the co-evolution of multiple MDD-based artifacts because of restrictions to express and propagate changes which manifest in (1) unidirectional model transformations and (2) disregarding transformation definitions as first-class models (see Fig. 1). An example for evolution mismatches is the inability to reflect changes in generated platform artifacts back to their corresponding instance models. For example, in Eclipse EMF, changes to the generated Java source code may be lost when executing the unidirectional M2T transformation once again. Furthermore, by default, the generation templates (i.e. Java Emitter Templates, JETs) cannot be easily adapted, excluding the possibility to reflect changes in the code generator logic (i.e. transformation definitions are not treated as first-class artifacts).

Yu et al. [26] provide a platform-specific (i.e. Java-bound) solution for the co-evolution problem stated above. In the approach, BX is used to synchronize models with generated and user-modified code. Prerequisites are that the platform-specific language encodes a textual duplicate of the PIM (i.e. `@model` annotations) and that a MM representation exists for the platform-specific language (i.e. a Java Ecore MM).

To establish BX, triple graph grammars [27] are commonly employed in MDD for keeping related models consistent (see, e.g., [28]). Triple graph transformations relate a source and a target graph (i.e. a model) by some correspondence graph. In this way, source and target graphs are coupled which provides a basic structure for their co-evolution.

Wachsmuth [29] considers MM/model co-evolution as a step-wise adaptation of MMs (via transformation relations) and instance-preservation of models. Instead of describing the co-evolution of models as a transformation between two MMs, Wimmer et al. [30] employ in-place transformations. Herrmannsdoerfer et al. [31] present a framework to model the co-evolution of MMs/models via the composition of coupled transactions to adapt the MM and specify the corresponding model migrations.

Furthermore, state-based MM/model co-evolution approaches, for instance, adopt HOTs which take a difference model obtained by comparing two MMs and generate a model transformation able to produce the co-evolution of involved models [9].

Although all of these co-evolution methods cope with model transformation restrictions, a combined and uniform solution is missing, so far. Either the approaches provide only one-way co-evolution possibilities (i.e. unidirectional) or only for a subset of MDD artifacts (e.g., only MM/model co-evolution). Therefore, in the next section, we propose a generic approach for co-evolving MDD-related artifacts.

3 Model Co-evolution via B-HOT

With our approach (Fig. 2 provides an overview) we want to overcome shortcomings of current methods and offer a generic solution for co-evolving MDD artifacts. The upper part of Fig. 2 reflects a traditional MDA. Model co-evolution is achieved by integrating (1) BX capabilities (lower part of Fig. 2) and (2) support for HOTs into the MDA.

As an example, consider a model transformation from an object-oriented representation (e.g., a class diagram) to a relational database model. For instance, both MMs are defined in a MOF-based language (see upper part of Fig. 2). Hence, their instance models (e.g., using Ecore as technological projection) conform to the (E)MOF MM. A transformation (e.g., specified via ATL or ETL) is

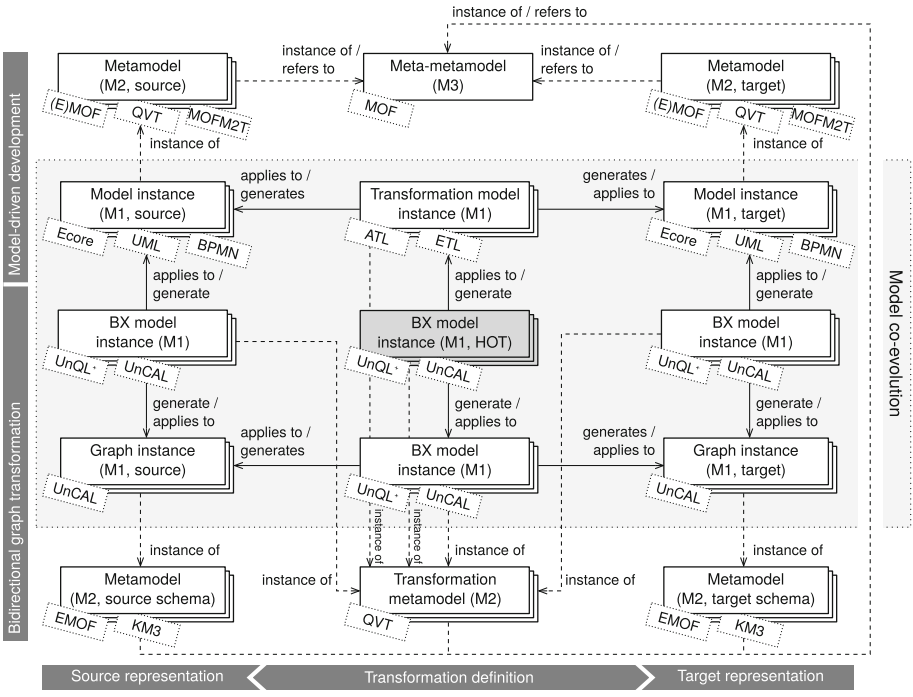

Fig. 2. Overview of our model co-evolution approach.

applied to class models and generates database models. This forward transformation proves useful in one case only: Changes in evolving class models should be reflected in database models, as well. Updates in database models cannot be propagated back to their source class models. A coupling of both representations limits the target model to be read-only (otherwise changes get lost when re-executing the transformation).

In our approach, we integrate BX capabilities by reusing native MDD concepts. *Every* transformation is represented as a first-class model conforming to a transformation MM. B-HOTs (see Fig. 2) provide for the mapping of unidirectional MDD-based transformation models (e.g. defined via ATL or ETL) into a bidirectional graph transformation model (and vice versa). Reconsidering the BX of the class-to-database model transformation example, in a next step, both source and target models (i.e. class and database models) are mapped to a graph structure (defined via UnCAL [32], a graph algebra). Source and target graph schemas are represented as MOF-based MMs. In order to establish bidirectionality between the class-to-database transformation, the unidirectional transformation model need to be mapped to a bidirectional transformation model. This is done via a B-HOT (defined via UnQL$^+$ [13], a SQL-like graph query/transformation language) which provides the forward and backward transformations between two transformation specifications (in our example, transformation models specified in ATL/ETL and UnQL$^+$). The result of the B-HOT is a BX specification (again defined via UnQL$^+$) which provides both, a forward transformation from class to database graphs as well as a corresponding backward transformation. Thus, changes in the database graph can be propagated back to the class graph. As the transformation of models to graphs is also bidirectional, updated class and database graphs can be represented in their initial model-based forms. Therefore, a BX of source and target models (class and database models in our example) is established. The backward database-to-class transformation is distinct to the BX and no corresponding MDD-based transformation equivalent exists (i.e. no backward transformation defined via, e.g., ATL or ETL). Therefore, as a last step, the backward transformation (in UnQL$^+$) must be represented in its original MDD-based form (in ATL, ETL) via the B-HOT mapping (see Fig. 2).

We discuss co-evolution properties of our approach according to the following four categories.

Model Relations: Our approach establishes BX-based relations between models, graphs, and model and graph representations. The mapping relation between traditional MDD-based transformations and BX representations (B-HOT) allows to add BX support in traditional MDAs. Furthermore, relations are not restricted to one source and one target artifact only, but can be used for the transformation of multiple dependent models/graphs, as well (see also compositional BX down below). The coupling of models via BX allows, on the one hand, to establish synchronization definitions and, on the other hand, to collect transformation traces. As many modeling artifacts make up a MDD process, keeping models consistent is of special importance. Moreover, trace information are a relevant source for documentation and debugging purposes.

Model Co-evolution Scenarios: Our approach supports any M2M relation and any number of MM-layers. Horizontal co-evolution examples are, for instance, the synchronization of different MMs or different instance models. Vertical co-evolution examples are, for instance, keeping instance models and corresponding MMs or PIMs and PSMs consistent. Consider, for example, a PIM representing a MM of an object-oriented system and a Java-based MM as its PSM equivalent. Both MMs are synchronized via a B-HOT keeping them consistent. In this case, changes in the PIM can be propagated into the PSM (and vice versa). If the Java-based MM needs to be modified (e.g., due to the release of a new Java version), these changes—when affecting the object-oriented system representation, as well—can be propagated back into the PIM via the B-HOT. Moreover, transformation models permit to propagate changes also in horizontal and vertical M2M and M2T transformation definitions, for instance, for the co-evolution of MMs and transformation models or different transformation models. Referring to the example of synchronizing a general object-oriented MM and a Java-based MM, consider that their instance models are transformed into a textual object-oriented representation and Java source code, respectively. Changes in the Java-based MM (again, e.g., because of a new Java version) must also be reflected in its M2T transformation (e.g., type changes). Via a B-HOT between the PIM and PSM M2T transformations, changes in one of the M2T transformations can be propagated into the other M2T transformation keeping them consistent.

Language-independent Integration: Our approach is not dependent on a specific model transformation language, i.e. it does not matter if the model transformation is defined via ATL, ETL, or any other language. This is because we do not integrate bidirectionality into a model transformation language directly. The B-HOT definition serves as a language-specific binding between the concepts of the unidirectional MDD-based transformation and the bidirectional graph transformation. These bindings must be specified only once for each MDD-based transformation language (e.g., ATL, ETL) and facilitate reuse of our approach.

BX Properties: We develop B-HOTs via a functional bidirectional graph transformation language named UnQL$^+$ [13]. UnQL$^+$ is an extension of UnQL [32], a graph querying language based on structural recursion (which can be expressed in first-order logic extended with transitive closure) [33]. The BX ensures the *well-behavedness* of forward and backward transformations (i.e. that they are consistent with each other) and satisfy the round-trip property [10]. As the BX does not restrict forward transformations to be information preserving, a backward transformation requires not only the modified target graph/model, but also the original source graph/model. Large BX can be developed in a *compositional* way of reusing existing information (e.g., via intermediate models). Compositional BX can be employed, on the one hand, for a pair of consecutive transformations, where the output of transformation A is the input of transformation B; for example, the output of the source model-to-graph transformation is fed into the forward source-to-target graph transformation (see Fig. 2). On the other hand, compositional BX can be used for a pair of transformations that share an identical input model, for instance, transformations from one PIM to multiple PSMs [12].

4 B-HOT Requirements

This paper provides a first step to make co-evolution in MDD via B-HOTs possible. Initial work regarding the methodology and accompanying tool support has been performed, but is far from being finalized. In this section, we list challenging requirements for the implementation of our approach. We present completed work and discuss prerequisites for future developments.

Transformation MMs: Our B-HOT approach relies on transformation (meta)models. MDD-based M2M transformation MMs exist, for instance, for ATL [11] and for a subset of the Epsilon-language family [34]. Regarding M2T transformations, Hoisl et al. [15] extended the Epsilon model representations of Wei [34]. The BX framework [13] does not need MM representations for the UnQL$^+$ and UnCAL languages. Syntax definitions in Backus-Naur Form (BNF) exist for both languages and need to be mapped to EMOF-compliant (e.g. Ecore-based) MM representations (ongoing work; see also Sects. 5 and 6).

MM-specific Bindings: For a B-HOT, language-specific bindings need to be established between uni- and bidirectional transformation MMs. Initial work provides a unidirectional transformation from a subset of the ATL language to UnQL$^+$ (excluding imperative code and OCL expressions of ATL). This transformation does not take the model representation of ATL into account [14][2]. Thus, language-specific bindings for, for instance, ATL and/or ETL to UnQL$^+$ and/or UnCAL via B-HOT is future work. Furthermore, a first prototype exists for the BX of model-to-graph representations (Ecore-based models to UnCAL and vice versa), but needs improvements (future work).

Round-tripping of Transformation Definitions: Our B-HOT approach demands transformation models as input. In contrast, most model transformation engines cannot execute model representations of transformation definitions. Therefore, the round-tripping of executable (i.e. text-based) transformation specifications and their model representations must be provided. For M2M transformations, "an ATL transformation is itself a model, conforming to the ATL MM" [11]. Furthermore, Wei [34] developed initial round-tripping support for an Epsilon subset which was extended for M2T transformations (i.e. EGL) by Hoisl et al. [15]. Currently, the automatically derived backward transformation of a BX can neither be expressed as UnQL$^+$ or UnCAL textual statements nor via corresponding model representations (future work).

Generic Mappings: Prototype developments define transformations in a specific language as implementation vehicle (e.g., ATL, ETL). To support uptake and transferability of our approach we need to establish mappings to OMG specifications (see also Fig. 2). Hoisl et al. [15] provide mappings between EGL-based M2T transformation concepts used for the prototype and the MOFM2T specification. As future work, uni-/bidirectional M2M transformation concepts (e.g., ATL, ETL, UnQL$^+$) will be mapped to the QVT specification.

[2] This separate work of integrating ATL and BX is performed in collaboration with the AtlanMod team, uses the same BX framework (*GRoundTram*), but in contrast focuses on unidirectional transformations from ATL to UnQL$^+$ with a concrete semantic alignment between these two technical spaces.

Development Support: Initial support for the requirements-driven testing of (meta)models and model transformations via scenarios is provided by Sobernig et al. [35] and is extended/evaluated by Hoisl et al. [36,37]. Furthermore, validation for source and target models as well as for BX is presented by Hidaka et al. [13]. We have started implementing an IDE (e.g., a text editor) to support the development of UnQL$^+$ BX (see Sect. 5) and UnCAL-based graphs (future work). For this task, we have chosen Eclipse Xtext as candidate framework because it combines the grammar specification for a textual syntax with an Ecore-based model representation and provides for an Eclipse-based IDE.

Combine BX and MDD: Model (i.e. graph) transformations are important to both BX and MDD [10,22]. Our approach allows to integrate BX into MDD, thereby reusing native methods and tools for both. We want to support the creation of a shared terminology [10] via the generalization and mapping of language-specific uni-/bidirectional transformation concepts to OMG specifications. After our developments have matured, as future work, we need to provide for a larger case study to show that our approach works in practice.

5 Prototypical MDD-Based Support for UnQL$^+$

This section introduces our initial MDD-based developments for the UnQL$^+$ BX language: an Ecore-based UnQL$^+$ MM (see Fig. 3), an Xtext grammar specification for the UnQL$^+$ textual syntax (see Listing 1), and editors to support the development of UnQL$^+$ BX in both textual and model-based syntax notations (see the example text- and model-based transformation definitions in Listing 2 and Fig. 4, respectively)[3] These developments are based on Eclipse EMF and matured versions will fulfill the following B-HOT requirements (see Sect. 4): The mapping of BNF-compliant/Xtext-based UnQL$^+$ grammar definitions to EMOF-compliant/Ecore-based MM representations. This allows the *specification of an UnQL$^+$ BX MM* with two corresponding and interchangeable concrete syntax variants (textual and model-based). As the Xtext grammar describes how an Ecore model is derived from a textual notation, *round-tripping of UnQL$^+$ BX definitions* is partially fulfilled (i.e. transformations specified textually are automatically mapped to their model representations). Furthermore, the implemented software artifacts (e.g., text/model editors) *support the development of UnQL$^+$ BX*.

Figure 3 shows an excerpt of the Ecore-based UnQL$^+$ MM. For this, the BNF grammar of UnQL$^+$ was mapped to a model representation consisting of (abstract) classes and class attributes as well as containment references and inheritance relationships between these classes. From Fig. 3 it can be seen that four different `Statement` types can be contained in an `UnQLplus` expression: `Selection`, `Replacement`, `Deletion`, and `Extension`. All of these statements operate on graphs (`Template`); for example, to select a graph based on certain conditions.

[3] All software artifacts can be obtained from the URLs mentioned in the footnote of Sect. 1.

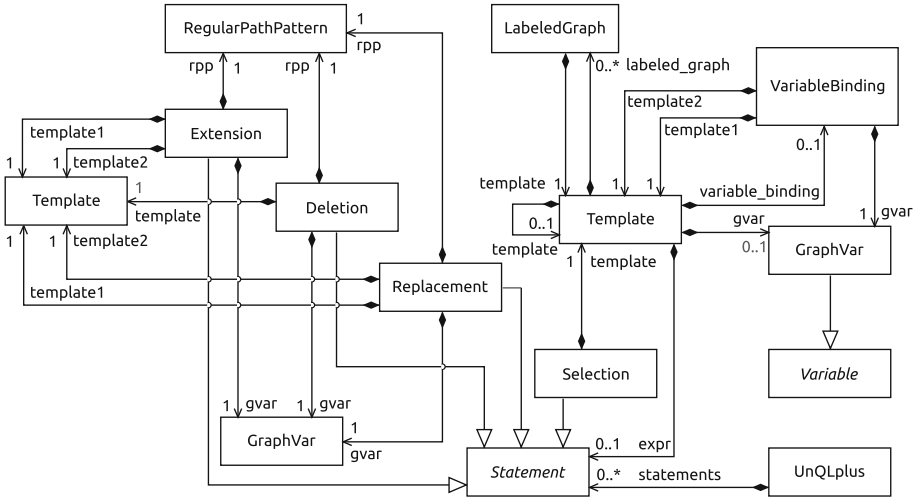

Fig. 3. Ecore-based UnQL$^+$ MM (excerpt).

Listing 1 shows an excerpt of the Xtext grammar for the UnQL$^+$ textual syntax. The grammar definition of UnQL$^+$ was translated from BNF to Xtext and aligned to match the Ecore model concepts. The entry rule on lines 7–8 in Listing 1 defines that an `UnQLplus` expression contains `Statements`. The `Statement` rule (lines 10–12) delegates in line 11 to the four alternative statement rules (defined later) and specifies the syntax of an optional condition (line 12). As examples, the syntax of two statements is specified in the `Selection` rule (lines 14–15) and in the `Replacement` rule (lines 17–18). Furthermore, the definition of graphs is shown in lines 22–23 (`Template` rule) and lines 25–33 (`TemplateExpression` rule), respectively (delegated rules are omitted).

```
1 grammar org.biglab.groundtram.bx.UnQLplus
2   hidden(WS, SL_COMMENT, ML_COMMENT)
3
4 import "http://unqlplus/0.1"
5 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
6
7 UnQLplus:
8   statements+=Statement*;
9
10 Statement:
11   (Selection | Replacement | Deletion | Extension)
12   ('where' condition+=Condition (',' condition+=Condition)*)?;
13
14 Selection:
15   'select' template=Template;
16
17 Replacement:
18   'replace' rpp=RegularPathPattern '->' gvar=GraphVar 'by' template1=
         Template 'in' template2=Template;
19
20 ...
```

```
21
22 Template:
23   TemplateExpression (=>'U' template=Template)?;
24
25 TemplateExpression returns Template:
26   {Template} '{' labeled_graph+=LabeledGraph? (',' labeled_graph+=
         LabeledGraph)* '}' |
27   '(' expr=Statement ')' |
28   fname=FunctionTemplate |
29   conditional=Conditional |
30   variable_binding=VariableBinding |
31   structural_recursion=StructuralRecursion |
32   mutual_structural_recursion=MutualStructuralRecursion |
33   gvar=GraphVar;
34
35 VariableBinding:
36   'let' gvar=GraphVar '=' template1=Template 'in' template2=Template;
37
38 ...
```

Listing 1. Xtext grammar definition for the UnQL$^+$ textual syntax (excerpt).

With the Ecore MM and the Xtext grammar defined in Fig. 3 and Listing 1, it is possible to provide editor support for textual as well as model-based UnQL$^+$ BX. Listing 2 shows an example class-to-database UnQL$^+$ BX replacing attributes by columns (excerpt taken from Hidaka et al. [13]). The BX in Listing 2 makes use of `Selection` (e.g., starting from line 1 and line 3) and `Replacement` statements (starting from line 7). The UnQL$^+$ BX was created using our Eclipse-based textual editor providing features, such as, syntax coloring, auto completion, error detection, and so forth. In this way, it is ensured that developed UnQL$^+$ BX conform to the Xtext grammar defined in Listing 1. A benefit of using the editor to write UnQL$^+$ BX is the early detection and immediate correction of syntactical errors.

```
1 select {tables : $table} where
2   $persistentClass in
3     (select $class where
4       {Association.(src|dest).Class : $class} in $db,
5       {is_persistent : {Boolean : true}} in $class),
6   $table in
7     (replace attrs -> $g
8       by (select {Column : $a} where
9          {attrs.Attribute : $a} in $persistentClass)
10      in $persistentClass)
```

Listing 2. Example class-to-database UnQL$^+$ BX (excerpt).

Figure 4 shows an excerpt of a tree-based view on a model representation of the same class-to-database UnQL$^+$ BX example introduced in Listing 2. Via the Ecore MM and the corresponding Xtext grammar, a model representation can be derived from textual UnQL$^+$ BX definitions. This instance model conforms to the Ecore-based UnQL$^+$ MM in the same way as a textual UnQL$^+$ BX definition conforms to the Xtext grammar. The representation of transformation models (e.g., expressing UnQL$^+$ BX as models as exemplified in Fig. 4) is one of the main requirements to realize our B-HOT approach.

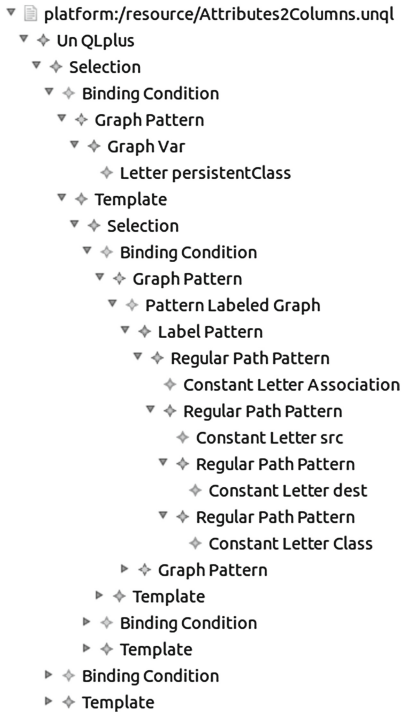

Fig. 4. Tree-based view on example class-to-database UnQL$^+$ BX model (excerpt).

6 Concluding Discussion

In this paper, we presented an approach for model co-evolution by combining BX and HOT for MDD. The developed method (B-HOT) intends to overcome current limitations for model co-evolution as transformations are represented as models and model transformations are bidirectionalized. In our approach, models are coupled via BX providing the benefit that synchronization of models is ensured via forward/backward transformations. Another advantage is that changes can be propagated into model transformations keeping them consistent with their evolving dependent artifacts (MMs, model instances). Our approach of integrating BX into MDD is generic and can be applied to any model transformation language via binding specifications.

With our work and according to the feature-based classification of BX approaches presented in [38], we combine graph-based and MDD-based artifact representations involved in BX. In particular, the employed demonstrator BX framework (GRoundTram) is based on graphs, while metamodels in Eclipse approximate the MOF specification (implemented via Ecore models). Therefore, the technical space of GRoundTram needs to cope with MDD-based artifact representations, as well. In GRoundTram (for further BX approaches and their characteristic features see [38]), the correspondence relation between source and

target artifacts is defined via user-provided unidirectional transformation specifications. In this case, the backward transformation is not explicitly stated (see also Sect. 4) and is automatically derived by the bidirectional engine (inversion). Thus, the results of the computed backward transformation needs to be conceptualized in the technical space of GRoundTram (i.e. represented in UnQL$^+$ or UnCAL). The proposed B-HOT approach presented in this paper demands to switch from an implicit to an explicit representation of the backward transformation.

Please note that BX approaches (see, e.g., [38]) differ in terms of feature characteristics and implementation methods, for example, according to the technical space (e.g., text-, graph-, MDD-based), the directionality of the consistency definition (e.g., unidirectional, bidirectional transformation specifications), the representation of changes (e.g., state-, operation-based), or the definition of the backward transformation (e.g., explicit, implicit) [38]. Although we focus in this paper specifically on one BX approach (i.e. GRoundTram), other BX methods which conceptually conform to the requirements discussed in the former paragraph as well as in Sect. 4 are candidates for consideration, as well.

A drawback of our proposal is that the efforts of creating initial modeling and transformation artifacts can be high. Transformation MMs may have to be defined for the intended target language. Currently, no bindings for transformation languages exist. Although these have to be defined only once for each language, this is a barrier for uptake. Transformation engines might not execute transformation models directly making model/text round-tripping functions necessary (but again these can be reused per language). Adequate tool support must be provided to facilitate the development of models and transformations.

Currently, we are developing an EMOF-based MM for the UnQL$^+$ BX language (in Ecore). In parallel, we transfer the BNF-based grammar definition to Xtext. This will ensure the consistent mapping of transformations written in UnQL$^+$ to their modeling equivalents. An editor to support the definition of UnQL$^+$ statements will be provided, as well. Initial developments (see also Sect. 5) are available at the URLs mentioned in the footnote of Sect. 1 and are continuously updated. UnQL$^+$ concepts will be mapped to the QVT-Relations language in the near future.

Acknowledgements. This work has partly been funded by the Austrian Research Promotion Agency (FFG) of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) through the Competence Centers for Excellent Technologies (COMET K1) initiative and the FIT-IT program.

References

1. Hoisl, B., Hu, Z., Hidaka, S.: Towards co-evolution in model-driven development via bidirectional higher-order transformation. In: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, pp. 466–471. SciTePress (2014)

2. Mellor, S., Clark, A., Futagami, T.: Model-driven development - guest editor's introduction. IEEE Softw. **20**, 14–18 (2003)
3. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, New York (2006)
4. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Softw. **20**, 42–45 (2003)
5. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**, 25–31 (2006)
6. Di Ruscio, D., Iovino, L., Pierantonio, A.: Coupled evolution in model-driven engineering. IEEE Softw. **29**, 78–84 (2012)
7. Meyers, B., Vangheluwe, H.: A framework for evolution of modelling languages. Sci. Comput. Program. **76**, 1223–1246 (2011)
8. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Softw. Syst. Model. **9**, 7–20 (2010)
9. Di Ruscio, D., Iovino, L., Pierantonio, A.: What is needed for managing co-evolution in MDE? In: Proceedings of the 2nd International Workshop on Model Comparison in Practice, pp. 30–38. ACM (2011)
10. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.F.: Bidirectional transformations: a cross-discipline perspective. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 260–283. Springer, Heidelberg (2009)
11. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., Bézivin, J.: On the use of higher-order model transformations. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 18–33. Springer, Heidelberg (2009)
12. Hidaka, S., Hu, Z., Kato, H., Nakano, K.: Towards a compositional approach to model transformation for software development. In: Proceedings of the 24th Symposium on Applied Computing, pp. 468–475. ACM (2009)
13. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Nakano, K.: GRoundTram: an integrated framework for developing well-behaved bidirectional model transformations. In: Proceedings of the 26th International Conference on Automated Software Engineering, pp. 480–483. IEEE (2011)
14. Sasano, I., Hu, Z., Hidaka, S., Inaba, K., Kato, H., Nakano, K.: Toward bidirectionalization of ATL with GRoundTram. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 138–151. Springer, Heidelberg (2011)
15. Hoisl, B., Sobernig, S., Strembeck, M.: Higher-order rewriting of model-to-text templates for integrating domain-specific modeling languages. In: Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development, pp. 49–61. SciTePress (2013)
16. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2nd edn. Addison-Wesley, Reading (2008)
17. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
18. Kolovos, D., Rose, L., García-Domínguez, A., Paige, R.: The Epsilon book (2015). http://www.eclipse.org/epsilon/doc/book/
19. Object Management Group: OMG meta object facility (MOF) core specification (2015). http://www.omg.org/spec/MOF, version 2.5, formal/2015-06-05
20. Object Management Group: Meta object facility (MOF) 2.0 query/view/transformation specification (2015). http://www.omg.org/spec/QVT, version 1.2, formal/2015-02-01
21. Object Management Group: MOF model to text transformation language (2008). http://www.omg.org/spec/MOFM2T, version 1.0, formal/2008-01-16

22. Hu, Z., Schurr, A., Stevens, P., Terwilliger, J.F.: Dagstuhl seminar on bidirectional transformations (BX). SIGMOD Rec. **40**, 35–39 (2011)
23. Bézivin, J., Gerbé, O.: Towards a precise definition of the OMG/MDA framework. In: Proceedings of the 16th International Conference on Automated Software Engineering, pp. 273–280. IEEE (2001)
24. Object Management Group: MDA guide (2003). http://www.omg.org/cgi-bin/doc?omg/03-06-01, version 1.0.1, omg/2003-06-01
25. OMG unified modeling language (OMG UML), superstructure (2015). http://www.omg.org/spec/UML, version 2.5, formal/2015-03-01
26. Yu, Y., Lin, Y., Hu, Z., Hidaka, S., Kato, H., Montrieux, L.: Maintaining invariant traceability through bidirectional transformations. In: Proceedings of the 34th International Conference on Software Engineering, pp. 540–550. IEEE (2012)
27. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
28. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. **8**, 21–43 (2009)
29. Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 600–624. Springer, Heidelberg (2007)
30. Wimmer, M., Kusel, A., Schönböck, J., Retschitzegger, W., Schwinger, W., Kappel, G.: On using inplace transformations for model co-evolution. In: Proceedings of the 2nd International Workshop on Model Transformation with ATL, INRIA & Ecole des Mines de Nantes (2010)
31. Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE - automating coupled evolution of metamodels and models. In: Drossopoulou, S. (ed.) ECOOP 2009. LNCS, vol. 5653, pp. 52–76. Springer, Heidelberg (2009)
32. Buneman, P., Fernandez, M., Suciu, D.: UnQL: a query language and algebra for semistructured data based on structural recursion. VLDB J. **9**, 76–110 (2000)
33. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Nakano, K.: GRoundTram: an integrated framework for developing well-behaved bidirectional model transformations. Prog. Inf. **10**, 131–148 (2013)
34. Wei, W.: EpsilonLabs: Epsilon static analysis (2012). http://code.google.com/p/epsilonlabs/wiki/EpsilonStaticAnalysis
35. Sobernig, S., Hoisl, B., Strembeck, M.: Requirements-driven testing of domain-specific core language models using scenarios. In: Proceedings of the 13th International Conference on Quality Software, pp. 163–172. IEEE (2013)
36. Hoisl, B., Sobernig, S., Strembeck, M.: Natural-language scenario descriptions for testing core language models of domain-specific languages. In: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, pp. 356–367. SciTePress (2014)
37. Hoisl, B., Sobernig, S., Strembeck, M.: Comparing three notations for defining scenario-based model tests: a controlled experiment. In: Proceedings of the 9th International Conference on the Quality of Information and Communications Technology, pp. 95–104. IEEE (2014)
38. Hidaka, S., Tisi, M., Cabot, J., Hu, Z.: Feature-based classification of bidirectional transformation approaches. Softw. Syst. Model. (2015)