A Compositional Approach to Bidirectional Model Transformation

Soichiro Hidaka Zhenjiang Hu Hiroyuki Kato
GRACE Center, National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430, Japan
{hidaka,hu,kato}@nii.ac.jp

Keisuke Nakano
The University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi
Tokyo 182-8585, Japan
ksk@cs.uec.ac.jp

Abstract

Bidirectional model transformation plays an important role in maintaining consistency between two models, and has many potential applications in software development, including model synchronization, round-trip engineering, software evolution, multiple-view software development, and reverse engineering. However, unclear bidirectional semantics, domain-specific bidirectionalization method, and lack of systematic development framework are known problems that prevent it from being practically used. In this paper, we propose a novel compositional framework for bidirectional model transformation based on an existing graph querying language UnQL, so that one can develop various useful bidirectional model transformation by combination of a fixed number of primitive bidirectional model transformations. We have implemented a prototype system, and the experimental results show promise of the new approach.

1 Introduction

Bidirectional model transformation [14, 2], being an enhancement of model transformation with bidirectional capability, is an important requirement on OMG's Queries/Views/Transformations (QVT) standard [13] recommended for defining model transformation languages. It describes 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. Bidirectional model transformation has many potential applications in software development, including model synchronization [2, 15, 7], round-trip engineering [1], software evolution by keeping different models coherent to each other [4], multiple-view software development [8, 6].

Despite these promising uses of bidirectional model transformation in software development, there are few serious practical applications in which bidirectional model transformation is used. One major problem, as strongly argued in [14], is that there lacks a clear definition of what bidirectional model transformation means. In general, a model transformation is not bijective, so a backward model transformation is much more involved than an inverse of forward model transformation. So in practice, without clear semantics of bidirectional transformation that ensures that backward transformation works correctly, no one would seriously use bidirectional transformation in such systems.

In this paper, we propose a novel *compositional* framework for bidirectional model transformation to solve this semantics problem, by integrating two state-of-the-art techniques: *functional* languages for bidirectional tree transformations [5, 12, 11] in the programming language community, and the *compositional* graph querying language UnQL [3] intensively studied in the database community. Our main result is that *bidirectional model (graph) transformations can be constructed by either a direct use of a primitive bidirectional model transformation, or a combination of smaller bidirectional model transformations to form a bigger one. Both the number of primitive bidirectional model transformations and the number of ways for combination are fixed.* This kind of compositionally is in sharp contrast to the existing approaches to (bidirectional) model transformation [2, 15, 7, 13] in that arbitrary number of intermediate models can be used in a model transformation.

2 Proposed Approach and Results

Figure 1 depicts an architecture (the basic idea) of our compositional framework. A model transformation is described in UnQL$^+$, an extension of an existing graph querying language UnQL [3], which is *functional* (rather than rule-based) and *compositional*. It is then desugared (translated) into a core *graph algebra* which consists of a set of simple graph constructors for building graphs and three combinators (sequential composition, condition, and structural recursion) for manipulating graphs. This graph alge-

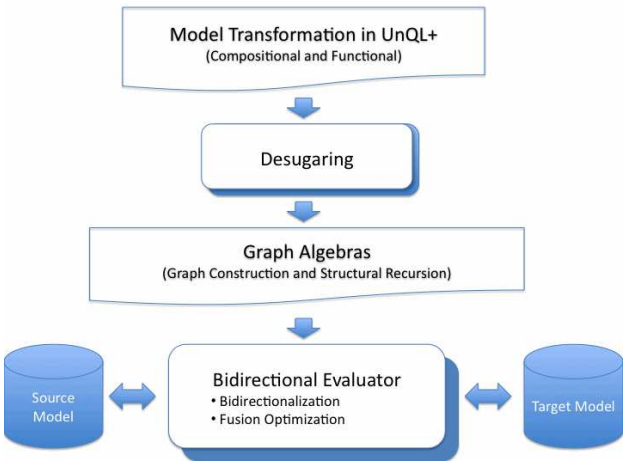

Figure 1. A Compositional Framework for Bidirectional Model Transformation Framework

bra can have clear bidirectional semantics and thus can be evaluated in a bidirectional manner.

Our main technical contributions can be summarized as follows.

- We made the first attempt of adapting an existing graph querying language for bidirectional model transformation, whose importance has not been recognized so far. We show that UnQL, a powerful graph querying language being suitable for systematic development of efficient graph queries, can be extended for systematic development of various bidirectional model transformations.
- We give a concise bidirectional semantics for the core algebra of UnQL$^+$. In particular, we show that structural recursion can be computed in a *bulk* way which is suitable for both forward and backward computation.
- We have implemented a prototype system in a functional language Objective Caml, and tested it with some non-trivial examples. The results are quite encouraging. We recommend the readers to take a look at the following website:

  http://www.biglab.org/

  to see more examples, play with the system, and find a technical report [9], a full version of this paper.

2.1 Models as Edge-labelled Graphs

Graphs in our framework follows those of UnQL, which are rooted and directed cyclic graphs with no order on outgoing edges. They are edge-labelled in the sense that all

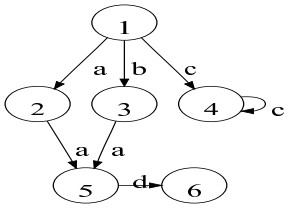

Figure 2. A Simple Graph

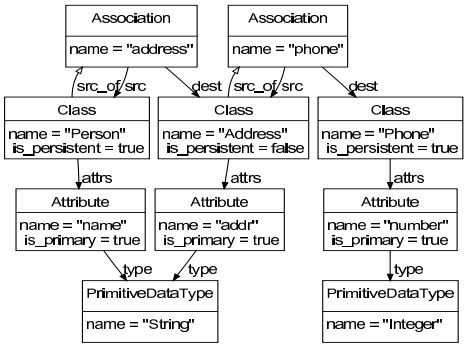

Figure 3. A Class Diagram

information is stored as labels on edges (the labels on nodes have no particular meaning). Figure 2 gives a small example of a directed cyclic graph with six nodes and seven edges. In text, it is represented by

$$\begin{array}{lcl} g & = & \{a : \{a : g_1\}, b : \{a : g_1\}, c : g_2\} \\ g_1 & = & \{d : \{\}\} \\ g_2 & = & \{c : g_2\} \end{array}$$

where the set $\{l_1 : e_1, \ldots, l_n : e_n\}$ denotes a graph which contains n edges with labels $l_1, \ldots, l_n$, each of which points to a graph again, and the empty set $\{\}$ denotes a graph with a single node. Two graphs can be merged using set union operation such as $g \cup g'$. As another example, the class model in Figure 3 consists of three classes and two directed associations, where each class has a primary attribute. This class model can be represented by the graph in Figure 4, where information is moved to edges.

2.2 Model Transformation in UnQL$^+$

UnQL [3] is a graph querying language based on structural recursion, and can be expressed using FO(TC) (first order with transitive closure), with time complexity of PTIME for graph querying. It, like other query languages, has a convenient and powerful *select-where* structure for extracting information from a graph. For instance, the following query extracts all persistent classes from the class model in

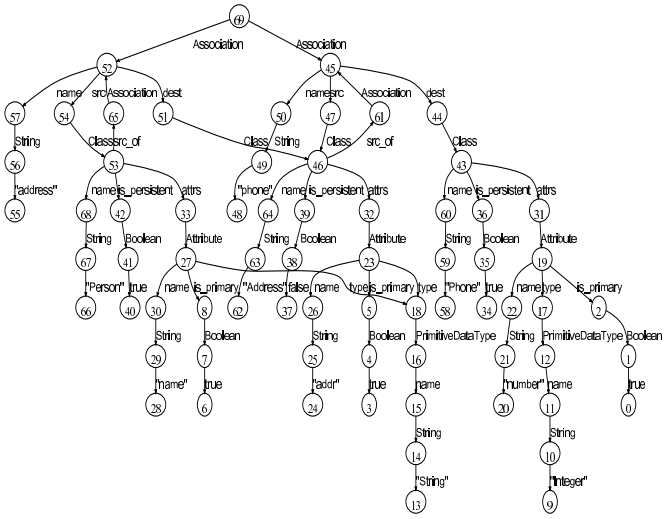

Figure 4. A Class Model Represented by an Edge-Labelled Graph

Figure 4, which is assumed to be bound by $\$classDB$.

select $\$class$ where
$\{Association.(src|dest).Class : \$class\}$ in $\$classDB$,
$\{is\_presistent : \{Boolean : true\}\}$ in $\$class$

The symbols prefixed with $ denote variables. This query returns all bindings of variable $\$class$ satisfying the two conditions in the where clause. The first condition is to find bindings of $\$class$ by matching the *regular path pattern* $Association.(src|dest).Class$ with the graph bound by $\$classDB$, while the second condition is to ensure that the class is persistent.

In model transformation, we often want to replace a subgraph satisfying certain condition by another graph. It is onerous to describe these kinds of graph transformations in UnQL because some context structure is required to be copied and propagated. UnQL$^+$ is an extension of UnQL with a new *replace-where* construct suitable for specifying model transformation [10]. We have already shown in [10] a typical but nontrivial model transformation, Class2-RDBMS, using UnQL$^+$. Here, we just touch UnQL$^+$ by using a simplified example. Consider, for example, that we want to add a prefix of $class\_$ to each class name in the class model. We may write it with the replace-where structure by

replace $\{\$name : \{\}\}$ by $\{("class\_" \hat{} \$name) : \{\}\}$ where
$\{\_*.Class.name.String : \{\$name : \{\}\}\}$ in $\$classDB$

where $\hat{}$ denotes string concatenation, and $\_*$ denotes arbitrary sequence of labels (in the path).

$$
\begin{array}{rcl}
E & ::= & \{\} \\
& | & \{L : E\} \\
& | & E \cup E \\
& | & \&x := E \\
& | & \&y \\
& | & () \\
& | & E \oplus E \\
& | & E \mathbin{@} E \\
& | & cycle(E) \\
& | & Var \\
& | & \mathsf{if}\ B\ \mathsf{then}\ E\ \mathsf{else}\ E \\
& | & rec(\lambda(LabelVar, Var).E)(Var) \\
& | & \mathsf{let}\ Var = E\ \mathsf{in}\ E
\end{array}
$$

Figure 5. UnCAL: A Graph Algebra

2.3 Desugaring UnQL$^+$ to Graph Algebra

To perform bidirectional computation of model transformations in UnQL$^+$, we transform UnQL$^+$ queries to expressions of UnCAL, a simple graph algebra, and then show how this algebra can be evaluated in a bidirectional way.

UnCAL [3], as defined in Figure 5, has a set of graph constructors and operators, by which arbitrary graphs can be represented and arbitrary graph transformation in UnQL$^+$ can be described [10]. The first nine expression structures are used to describe very primitive transformations for graph construction, while the last three are combinators, namely condition, structural recursion and sequential composition, for composing smaller transformations to form a bigger one. We omit the detailed discussion of UnCAL which can be found in [3].

2.4 Bidirectional Evaluator

UnCAL can be bidirectionalized in the sense that a formal and sound bidirectional semantics can be given to UnCAL, such that the forward computation performs the same as the usual UnCAL evaluator does, and the backward computation is guaranteed to satisfy the bidirectional properties [5] with respect to the forward computation.

More specifically, given an expression e and an environment ρ denoting a mapping from variables to values (a label or a graph), we define two computations: a *forward computation* $\rho \overset{e}{\rightharpoonup} g$ is to evaluate the expression e to a graph g under the environment ρ, while a *backward computation* $\rho' \overset{e}{\leftharpoonup}_{\rho} g'$ is to compute a new environment ρ' from the old ρ and a revised graph g' over g obtained from forward computation. The forward and backward computations with respect to an expression e should satisfy the following two properties [5]: (1) The *GetPut* Property: no change on the graph should give no change on the environment, i.e., $\rho \overset{e}{\rightharpoonup} g$ implies $\rho \overset{e}{\leftharpoonup}_{\rho} g$; (2) The *PutGet*

Property: the backward computation computes a new environment ρ' from g' in such a way that applying the forward computation under ρ' again should give the same graph g', i.e., $\rho' \overset{e}{\leftarrow}_{\rho} g'$ implies $\rho' \overset{e}{\rightarrow} g'$.

For bidirectionalization of UnCAL, since UnCAL is compositional, consisting nine primitive transformations and three combinators, it suffices to give forward and backward computations for nine primitive transformations, and show how to obtain forward and backward computation for the three combinators from that of smaller transformations used in them. This is exactly the approach successfully used in bidirectional tree transformations [5, 12, 11]. As a simple example, the following two rules showing how to deal with forward and backward computation for graph union $e_1 \cup e_2$ respectively.

$$\frac{\rho \overset{e_1}{\rightarrow} g_1 \quad \rho \overset{e_2}{\rightarrow} g_2}{\rho \overset{e_1 \cup e_2}{\rightarrow} g_1 \cup g_2} \text{(FWD)}$$

$$\frac{g' \Rightarrow (g'_1, g'_2) \quad \rho_1 \overset{e_1}{\leftarrow}_{\rho} g'_1 \quad \rho_2 \overset{e_2}{\leftarrow}_{\rho} g'_2}{\rho_1 \uplus_{\rho} \rho_2 \overset{e_1 \cup e_2}{\leftarrow}_{\rho} g'} \text{(BWD)}$$

The rule FWD says that two graphs g_1 and g_2 are computed from e_1 and e_2 under the environment ρ respectively, and give a result as a union of the two graph. And the rule BWD says that the possibly updated output graph g' is decomposed into two parts so that backward computations for e_1 and e_2 can be successfully performed resulting in two new environments ρ_1 and ρ_2, and the result environment is a disjoint union of the two environments. We leave other detailed discussion in [9], and concrete example in a demo system available at http://www.biglab.org/.

3 Impact of Results and Future Work

As far as we are aware, we made the first attempt of designing and implementing a functional language UnQL$^+$ for bidirectional model transformations, which is different from the existing rule-based (relational) approaches. We demonstrate that functional approach is helpful to give bidirectional semantics in a formal and concise way. This work is our first step towards *bidirectional model programming*, a linguistic framework to support systematic development of model transformations. We wish to look more into its relation with the rule-based approaches, and see how to combine them to form a more powerful framework for bidirectional model transformation.

References

[1] M. Antkiewicz and K. Czarnecki. Framework-specific modeling languages with round-trip engineering. In *MoDELS 2006: Proceedings of the 9th nternational Conference on Model Driven Engineering Languages and Systems*, pages 692–706. Springer-Verlag, 2006.

[2] M. Antkiewicz and K. Czarnecki. Design space of heterogeneous synchronization. In *GTTSE '07: Proceedings of the 2nd Summer School on Generative and Transformational Techniques in Software Engineering*, 2007.

[3] P. Buneman, M. F. Fernandez, and D. Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *VLDB Journal: Very Large Data Bases*, 9(1):76–110, 2000.

[4] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information preserving bidirectional model transformations. In *Fundamental Approaches to Software Engineering*, pages 72–86. 2007.

[5] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *POPL '05 : ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 233–246, 2005.

[6] M. Garcia. Bidirectional synchronization of multiple views of software models. In *Proceedings of DSML-2008*, volume 324 of *CEUR-WS*, pages 7–19, 2008.

[7] H. Giese and R. Wagner. Incremental model synchronization with triple graph grammars. In *MoDELS 2006: Proceedings of the 9th nternational Conference on Model Driven Engineering Languages and Systems*, pages 543–557. Springer Verlag, 2006.

[8] J. Grundy, J. Hosking, and W. B. Mugridge. Inconsistency management for multiple-view software development environments. *IEEE Trans. Softw. Eng.*, 24(11):960–981, 1998.

[9] S. Hidaka, Z. Hu, H. Kato, and K. Nakano. An algebraic approach to bidirectional model transformations. Technical Report GRACE-TR08-02, GRACE Center, National Institute of Informatics, Sept. 2008.

[10] S. Hidaka, Z. Hu, H. Kato, and K. Nakano. Towards a compositional approach to model transformation for software development. In *24th Annual ACM Symposium on Applied Computing (SAC 2009) (to appear)*, Mar. 2009.

[11] Z. Hu, S.-C. Mu, and M. Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. *Higher-Order and Symbolic Computation*, 21(1-2):89–118, 2008.

[12] K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and a. Masato Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *12th ACM SIGPLAN International Conference on Functional Programming (ICFP 2007)*, pages 47–58. ACM Press, Oct. 2007.

[13] OMG. MOF QVT final adopted specification. `http://www.omg.org/docs/ptc/05-11-01.pdf`, 2005.

[14] P. Stevens. Bidirectional model transformations in QVT: Semantic issues and open questions. In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *Proc. 10th MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[15] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei. Towards automatic model synchronization from model transformations. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, pages 164–173. ACM Press, Nov. 2007.