GRoundTram: An Integrated Framework for Developing Well-Behaved Bidirectional Model Transformations

Soichiro Hidaka*, Zhenjiang Hu*, Kazuhiro Inaba*§,
Hiroyuki Kato* and Keisuke Nakano†
*National Institute of Informatics, Japan
Email: {hidaka,hu,kinaba,kato}@nii.ac.jp
†University of Electro-Communications, Japan
Email: ksk@cs.uec.ac.jp

Abstract— **Bidirectional model transformation is useful for maintaining consistency between two models, and has many potential applications in software development including model synchronization, round-trip engineering, and software evolution. Despite these attractive uses, the lack of a practical tool support for systematic development of well-behaved and efficient bidirectional model transformation prevents it from being widely used. In this paper, we solve this problem by proposing an integrated framework called GRoundTram, which is carefully designed and implemented for compositional development of well-behaved and efficient bidirectional model transformations. GRoundTram is built upon a well-founded bidirectional framework, and is equipped with a user-friendly language for coding bidirectional model transformation, a new tool for validating both models and bidirectional model transformations, an optimization mechanism for improving efficiency, and a powerful debugging environment for testing bidirectional behavior. GRoundTram has been used by people of other groups and their results show its usefulness in practice.**

I. Introduction

Bidirectional model transformation [1–5], being an enhancement of model transformation with bidirectional capability, is an important requirement in OMG's Queries/Views/Transformations (QVT) standard [6]. It describes not only a forward transformation from a source model to a target model, but also a backward transformation showing how to reflect the changes in the target model to the source model so that consistency between two models is maintained. Unlike (unidirectional) model transformation where lots of tools have been developed for supporting design, validation, and test of model transformation, bidirectional model transformation lacks such useful tools, which prevents it from being widely used. In fact, new requirements and challenges are introduced in the context of bidirectional model transformation.

First and most important, we should be sure that the bidirectional model transformation behaves exactly as we want. Unlike unidirectional model transformation, bidirectional model transformation has more complicated behavior than unidirectional one. It should be *well-behaved* in the sense that both forward and backward transformations are consistent with each other and satisfy the roundtrip property [1]. As argued in [2], there exist semantic issues in many existing tools.

Next, bidirectional model transformations should be *compositional* to reuse existing transformations and construct bigger ones from smaller ones. As indicated in the conclusion in [7], most model transformation languages based on graph transformations are rule-based, describing direct relationship between the source and target models. They are not compositional in the sense that we cannot introduce *intermediate models* for gluing model transformations. This makes them hard to support systematic development of model transformations in the large [8]. However, composition comes at the cost of efficiency; many unnecessary intermediate models might be produced. Therefore, an optimization method are required to automatically eliminate unnecessary intermediate models during execution.

Furthermore, bidirectional model transformation should be general enough as it is used at various stages of software development life cycle. It is applied to different models such as UML diagrams, sequence diagrams, Petri-nets, and even lower level control/data flow graphs. While visual frameworks are useful in high level design, *general text-based languages* play an important role in developing large-scale transformations, say, to deal with lower level mapping or complex code refactoring. Besides, we would expect to have a set of language-based tools for type checking (validating) both models and bidirectional model transformations to remove unnecessary errors before execution, an efficient execution model, and a tool for testing/debugging bidirectional behavior. We believe that such *language-based modeling environments* play important role in bidirectional model transformation.

In this paper, we remedy this situation by proposing a language-based modeling framework called GRoundTram [9], which is carefully designed and implemented for *compositional* development of *well-behaved* and *efficient* bidirectional model transformation at various stages of software

§Current affiliation is Google Inc. Email: kiki@kmonos.net

development. Our work is greatly inspired by recent research on bidirectional languages and automatic bidirectionalization in the programming language community [10–13]. In particular, it has been recently shown [14] that a graph query algebra UnCAL can be fully bidirectionalized. Each graph transformation in UnCAL has a clear bidirectional semantics and is guaranteed to be well-behaved.

This paper is about a successful application of the result of a bidirectional graph query algebra in the programming community to the construction of a framework for developing bidirectional model transformation in the software engineering community. Our main technical contributions are summarized as follows.

- **Well-Behavedness**. We propose a novel *bidirectional graph contraction algorithm* so that we can build well-behaved bidirectional model transformations upon the well-founded bidirectional UnCAL algebra. In fact, there is a gap between the UnCAL graphs and the models in model transformation: graphs in UnCAL are edge-labeled and their equality is defined by bisimulation, while models in model transformation may have labels on both edges and nodes and their equality is defined by unique identifiers. We close this gap so that every UnCAL graph has a bidirectional correspondence with a model.
- **Compositional**. We design a UnQL$^+$, a *purely functional languages* for developing large bidirectional model transformations in a *compositional* way. UnQL$^+$ is an extension of the graph query language UnQL [15] with new additional language constructs for graph transformation. We show that any UnQL$^+$ program can be correctly translated to an UnCAL construct and inefficiency due to intermediate models in the composition can be automatically eliminated.
- **Languages-based IDE**. We implement an integrated development environment GRoundTram, which has a novel tool for validating both models and bidirectional model transformations, an automatic optimization mechanism for improving efficiency, and a powerful debugging environment for testing bidirectional behavior. The system (including the sources, the documents, and many application examples) is available online [9], and has been and is being used by people of other groups for developing some nontrivial applications. Their results indicate its usefulness in practice.

II. Overview of GRoundTram

Figure 1 shows the basic functions the GRoundTram system provides.

A. Input

The input to the system is a source model together with its schema, a transformation described in UnQL$^+$, and a target model schema. The target model is produced by the forward transformation.

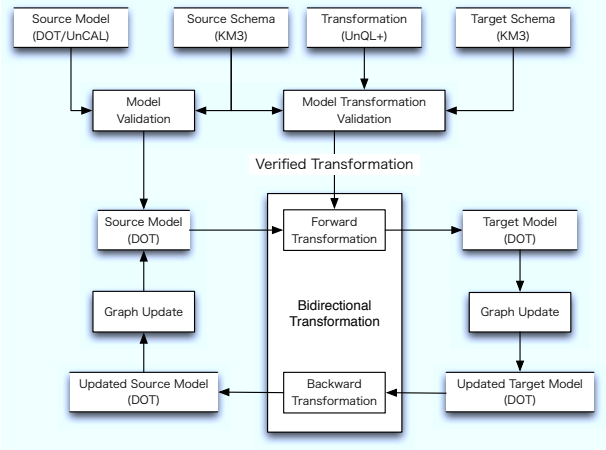

Figure 1. Overview of GRoundTram

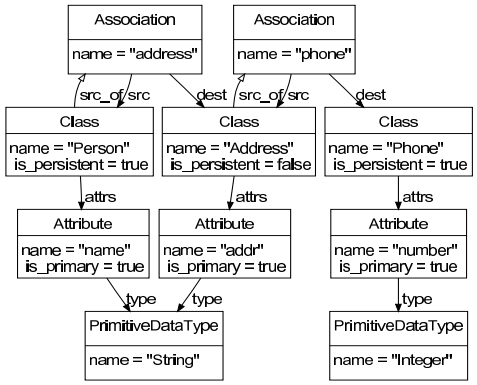

Figure 2. A Class Diagram

- **Model**. Models are represented by general edge-labeled graphs, which form a general representation of various models. As a concrete example, consider the class model diagram in Figure 2. It consists of three classes and two directed associations, and each class has a primary attribute. This model can be represented by the graph where information is moved from nodes to edges.
- **Model Schema (Metamodel)**. Each model has a structure. For instance, a class diagram has the following structure. A class diagram consists of classes and directed associations between classes. A class is indicated as persistent or non-persistent. It consists of one or more attributes, at least one of which must be marked as constituting the classes' primary key. An attribute type is of a primitive data type (e.g. String, Integer). An association specifies an inheritance relation between two classes. KM3 [16] is used to describe such a model structure, and its definition can be found in [9].

- **Model Transformation**. (Forward) Model transformation is described compositionally in UnQL$^+$, a SQL-like graph query/transformation language. As an example, consider extracting all persistent classes from the class model $\$db$, and transforming them to tables by replacing *Attributes* by *Columns*. This can be described compositionally as follows, where the intermediate model $\$persistentClass$ is used in this composition.

$$
\begin{array}{l}
\textbf{select}\{tables : \$table\}\ \textbf{where} \\
\quad \$persistentClass\ \textbf{in} \\
\quad\quad \text{(* select classes *)} \\
\quad\quad (\textbf{select}\ \$class\ \textbf{where} \\
\quad\quad\quad \{Association.(src|dest).Class : \$class\}\ \textbf{in}\ \$db, \\
\quad\quad\quad \{is\_persistent : \{Boolean : true\}\}\ \textbf{in}\ \$class), \\
\quad \$table\ \textbf{in} \\
\quad\quad \text{(* replace } Attribute \text{ *)} \\
\quad\quad (\textbf{replace}\ attrs \rightarrow \$g \\
\quad\quad\quad \textbf{by}\ \ (\textbf{select}\{Column : \$a\}\ \textbf{where} \\
\quad\quad\quad\quad \{attrs.Attribute : \$a\}\ \textbf{in}\ \$persistentClass) \\
\quad\quad\quad \textbf{in}\ \$persistentClass)
\end{array}
$$

B. Validation

In order to detect errors during development as early as possible and help users to develop a correct models and transformations, the GRoundTram system provides two types of validation mechanisms.

- **Model Validation**. Conformance of the source and the target model to their associated schemas can be verified by the system. In particular after editing the models, it is important to check that they are in valid states.
- **Model Transformation Validation**. Correct model transformations should always generate a target model conforming to the target schema from any source model satisfying the source schema.

While the model validation is quite standard, a general model transformation validation is more challenging but more useful in developing correct model transformation. As an instance of simple erroneous transformation, suppose the user made an error writing **select** $\$a$ instead of **select** $\{Column : \$a\}$ in the previous example. Its outputs do not conform to the schema and hence reported by the system. The check is *automatic* and *static*. Users neither have to provide any test cases by hand nor execute the transformation for testing; the system automatically finds out and displays an example of a source model that reveals the problem (in this case, a class model containing at least one persistent class).

C. Bidirectional Transformation

The GRoundTram system is unique in its execution of well-behaved bidirectional transformation, as seen in the lower part of Figure 1.

- **Forward Transformation**. After the user specified the source model and the UnQL$^+$ model transformation, by running the transformation with the model set to $\$db$ variable, the target model is computed. Like the source model, the target model can also be exported in the standard DOT format and be edited.
- **Backward Transformation**. The most distinct feature of GRoundTram is the automatic derivation of backward transformations that appropriately propagate modifications on target models to source graphs. There is no need to maintain two separate transformations and to worry about their consistency. Users just write a forward transformation from one model to another in a compositional way, and a corresponding backward transformation is automatically derived.

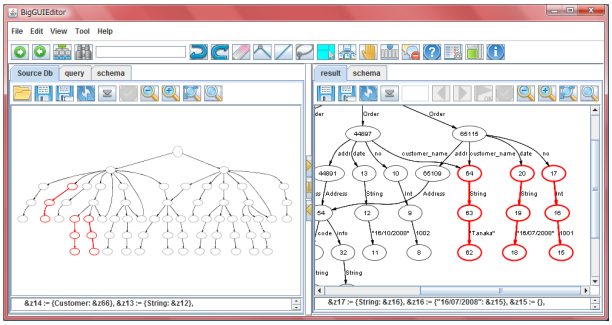

Figure 3. GUI of GRoundTram

III. Implementation of GRoundTram

A. Graphic User Interface

The GRoundTram system combines all the functions as an integrated framework with a user-friendly GUI (Figure 3). The user loads a source graph (displayed in the left pane) and a bidirectional transformation written in UnQL$^+$. Once they are loaded, forward transformation can be conducted by pushing the "forward" button (right arrow icon). The target graph appears on the right pane. User can graphically edit the target graph and apply backward transformation by pushing the "backward" button (left arrow icon). Source graph can be edited as well, of course. User can optionally specify the source schema and the target schema, and can run validation by pushing the check button on both panes. The transformation itself can also be checked.

For ease of debugging/understanding behavior of bidirectional computation between two models, *trace information* is instantly displayed between source and target (red part in Figure 3). If subgraphs on either pane are selected, corresponding subgraphs on the other pane are also highlighted. This helps users to understand how modification on the target affects that on the source, and vice versa.

B. System Architecture

Figure 4 depicts the architecture of the system. We provide a new model transformation language UnQL$^+$, and we ac-

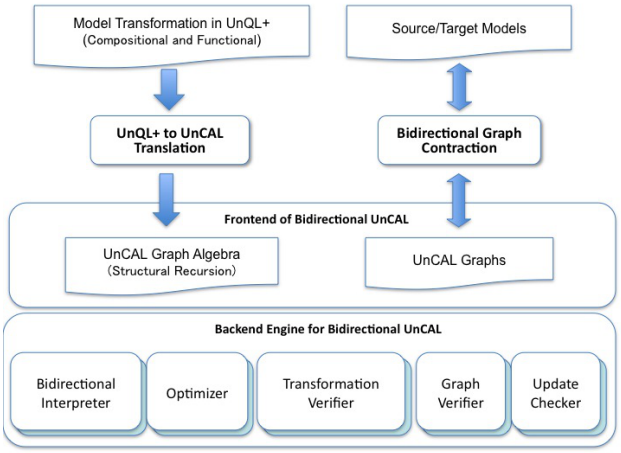

Figure 4. GRoundTram Implementation on Bidirectional UnCAL Engine

cept models that are described by edge-labeled graphs which are general enough to capture various kinds of models. We implement the GRoundTram system upon the powerful engine of bidirectional UnCAL, where a set of language-based tools have been developed: a bidirectional interpreter [14], a graph and graph transformation verifier [17], an optimizer to improve efficiency [18], and a checker of valid updates in the backward transformation [19]. The key contributions in this implementation are (1) a translation of UnQL$^+$ to UnCAL to enable use of the engine of bidirectional UnCAL, and (2) a bidirectional graph contraction algorithm for contracting bisimilar UnCAL graphs so that a usual model can have a bidirectional correspondence with an UnCAL graph.

IV. Concluding Remarks

The GRoundTram system has been fully implemented and is available online. The system website [9] provides a bunch of examples, big and small, including the known nontrivial (bidirectional) model transformation between UML class diagrams and relational databases.

This work is our first step towards *bidirectional model programming*, a linguistic framework to support systematic development of model transformation programs. In the future, we wish to look more into relation between the rule-based approach and the algebraic and functional approach, and see how to integrate them to have a more powerful framework for bidirectional model transformation.

References

[1] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger, "Bidirectional transformations: A cross-discipline perspective," in *International Conference on Model Transformation (ICMT 2009)*. LNCS 5563, Springer, 2009, pp. 260–283.

[2] P. Stevens, "Bidirectional model transformations in QVT: Semantic issues and open questions," in *Proc. 10th MoDELS*, ser. Lecture Notes in Computer Science, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, Eds., vol. 4735. Springer, 2007, pp. 1–15.

[3] ——, "A landscape of bidirectional model transformations," in *Generative and Transformational Techniques in Software Engineering II*, R. Lämmel, J. Visser, and J. a. Saraiva, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 408–424.

[4] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer, "Information preserving bidirectional model transformations," in *Proceedings of the 10th international conference on Fundamental approaches to software engineering*, ser. FASE'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 72–86.

[5] 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.

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

[7] K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, U. Prange, G. Taentzer, D. Varró, and S. Varró-Gyapay, "Model transformation by graph transformation: A comparative study," in *MTiP 2005, International Workshop on Model Transformations in Practice*. Springer-Verlag, 2005.

[8] F. Klar, A. Königs, and A. Schürr, "Model transformation in the large," in *ESEC/SIGSOFT FSE*, I. Crnkovic and A. Bertolino, Eds. ACM, 2007, pp. 285–294.

[9] "The BiG project web site," http://www.biglab.org/.

[10] 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*, 2005, pp. 233–246.

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

[12] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt, "Boomerang: resourceful lenses for string data," in *POPL '08: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, G. C. Necula and P. Wadler, Eds. ACM, 2008, pp. 407–419.

[13] Z. Hu, S.-C. Mu, and M. Takeichi, "A programmable editor for developing structured documents based on bidirectional transformations," *Higher-Order and Symbolic Computation*, vol. 21, no. 1-2, pp. 89–118, 2008.

[14] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, and K. Nakano, "Bidirectionalizing graph transformations," in *ACM SIGPLAN International Conference on Functional Programming*. ACM, 2010, pp. 205–216.

[15] 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*, vol. 9, no. 1, pp. 76–110, 2000.

[16] F. Jouault and J. Bézivin, "KM3: A DSL for metamodel specification," in *Formal Methods for Open Object-Based Distributed Systems*. LNCS 4037, Springer, 2006, pp. 171–185.

[17] K. Inaba, S. Hidaka, Z. Hu, H. Kato, and K. Nakano, "Graph-transformation verification using monadic second-order logic," in *Proceedings of the 13th ACM SIGPLAN international conference on Principles and practice of declarative programming (PPDP 2011)*, Odense, Denmark, 2011.

[18] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, K. Nakano, and I. Sasano, "Marker-directed Optimization of UnCAL Graph Transformations," in *Proceediings of 21st International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2011)*, Odense, Denmark, 2011.

[19] K. Nakano, S. Hidaka, Z. Hu, K. Inaba, and H. Kato, "Simulation-based graph schema for view updatability checking of graph queries," GRACE Center, National Institute of Informatics, Tech. Rep. GRACE-TR11-01, May 2011.