ISSN 1884-0760

GRACE TECHNICAL REPORTS

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

Soichiro Hidaka Zhenjiang Hu Kazuhiro Inaba
Hiroyuki Kato Keisuke Nakano

GRACE-TR 2011–05 August 2011

CENTER FOR GLOBAL RESEARCH IN
ADVANCED SOFTWARE SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF INFORMATICS
2-1-2 HITOTSUBASHI, CHIYODA-KU, TOKYO, JAPAN

WWW page: http://grace-center.jp/

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 novel 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 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 showing how to reflect the changes in 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 [5–7], round-trip engineering [8], software evolution [9], and multiple-view software development [10,11].

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, we have to introduce new requirements and challenges 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. 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 [12], 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 [13]. 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. As far as we are aware, no such *language-based modeling environments* have been proposed for bidirectional model transformation.

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

In this paper, we remedy this situation by proposing a language-based modeling framework called GRoundTram, which is carefully designed and implemented for *compositional* development of *well-behaved* and *efficient* bidirectional model transformation at various stages of software development. Our work is greatly inspired by recent research on bidirectional languages and automatic bidirectionalization in the programming language community [14–17]. In particular, it has been recently shown [18] 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 user-friendly language UnQL$^+$, which is the first *purely functional languages* for developing large bidirectional model transformations in a *compositional* way. UnQL$^+$ is an extension of the graph query language UnQL [19] 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 [20], and has been and is being used by people of other groups for developing some nontrivial applications. Their results indicate its usefulness in practice.

The rest of the paper is organized as follows. We begin by demonstrating how the GRoundTram system works in Section II. Then we briefly review the UnCAL bidirectional framework on which GRoundTram is based in Section III. After explaining the design architecture of the GRoundTram system, we show in details the definition of UnQL$^+$, the bidirectional graph contraction algorithm, and the translation from UnQL$^+$ to UnCAL in Section IV. We evaluate the system in Section V, discuss the related work in Section VI, and conclude the paper in Section VII.

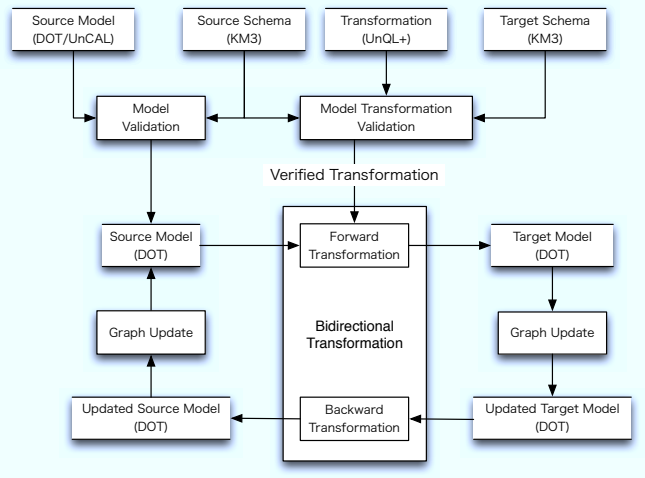


Figure 1. Overview of GRoundTram

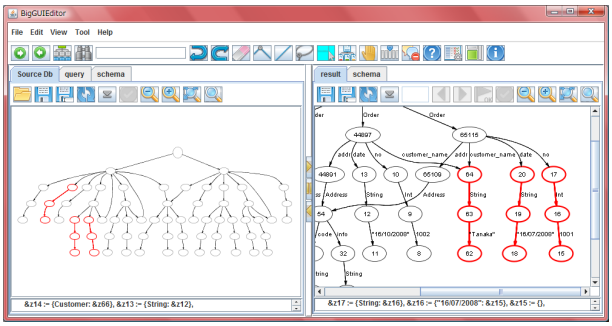


Figure 2. Snapshot of GRoundTram

II. Overview of GRoundTram

Before proceeding with the technical details, we give a brief overview of the GRoundTram system to give a flavor of what it can do. 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.

- **Model**. Models are represented by general edge-labeled graphs, which form a general representation of various

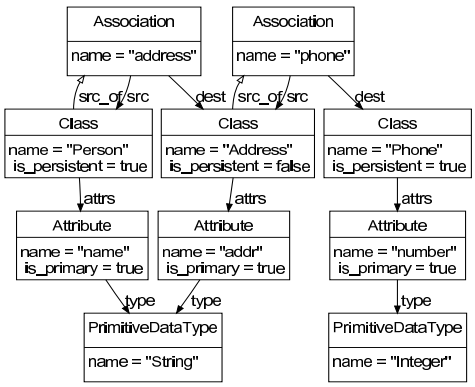


Figure 3. A Class Diagram

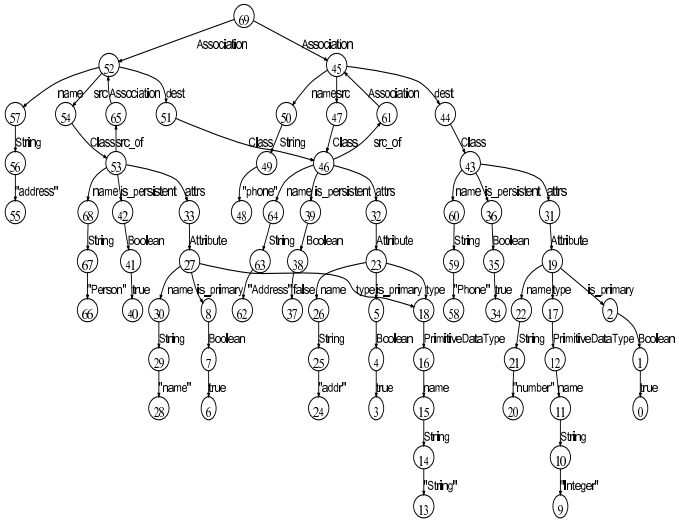


Figure 4. A Class Model Represented by an Edge-Labelled Graph

models. As a running example, consider the class model diagram in Figure 3. It consists of three classes and two directed associations, and each class has a primary attribute. This model can be represented by the graph in Figure 4, where information is moved to edges. The graph is in the standard DOT format which can be visualized and edited by the popular Graphviz tool [21].

- **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 [22] is used to describe such a model structure, and its definition can be found in [20].
- **Model Transformation**. (Forward) Model transformation is described compositionally in UnQL$^+$ (Section IV-A), 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 $\textbf{select}\ \$a$ instead of $\textbf{select}\ \{Column : \$a\}$ 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.

D. Graphic User Interface

The GRoundTram system combines all the functions as an integrated framework with a user-friendly GUI (Figure 2). 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.

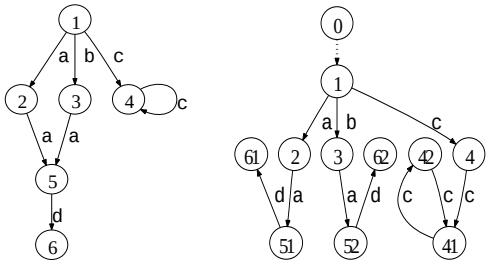
For ease of debugging/understanding behavior of bidirectional computation between two models, *trace information* is instantly displayed between source and target (red part in Figure 2). If subgraphs on either pane are selected, corresponding subgraphs on the other pane are also highlighted. This helps users to understand how modification on the target affects that on the source, and vice versa.

III. Background: Bidirectional UnCAL

The GRoundTram system is built upon the recent work [18] on bidirectionalization of UnCAL, a graph algebra known in the database community for graph querying [19]. It has been shown that any unidirectional graph transformation written in UnCAL can be fully bidirectionalized with a backward transformation such that both forward and backward transformations are consistent and well-behaved. We briefly explain the basic results that will be used in this paper.

A. Graph Data Model

Graphs in UnCAL are rooted and directed cyclic graphs with no order between outgoing edges. They are edge-labeled in the sense that all information is stored as labels on edges and labels on nodes serve as unique identifiers and have no particular meaning. Figure 5(a) gives a small



(a) A Simple Graph (b) An Equivalent Graph

Figure 5. Graph Equivalence Based on Bisimulation

example of a directed cyclic graph with six nodes and seven edges. In text, it is represented by

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

where the notation $\{l_1 : g_1, \ldots, l_n : g_n\}$ denotes a set representing a graph which contains n edges with labels $l_1, \ldots, l_n$, each edge pointing to a graph g_i, and the empty set $\{\}$ denotes a graph with a single node. Two graphs g_1 and g_2 can be merged using set union operation $g_1 \cup g_2$. In addition, the ε-edge is allowed to represent shortcut of two nodes, and works like ε-transition in automata.

Two graphs in UnCAL are considered to be equal if they are *bisimilar*. Intuitive understanding of bisimulation is that unfolding of cycles and duplication of equivalent subgraphs do not affect equivalence of graphs, and unreachable parts from the root are ignored. For instance, the graph in Figure 5(b) is equivalent to the graph in Figure 5(a); the new graph has an additional ε-edge (denoted by the dotted line), duplicates the graph rooted at node 5, and unfolds and splits the cycle at node 4.

It is worth noting that bisimulation equivalence plays an important role in bidirectionalization [18], query optimization [19], and verification of graph transformation [23]. However, bisimulation equivalence is different from our usual equivalence of models whose element has a unique identifier. We will show how to bridge this gap in Section IV-B.

B. UnCAL

The most important feature of UnCAL is that any graph transformation in UnCAL is described by structural recursions or their composition.

Structural recursive function f in UnCAL is a recursive computation scheme on graphs defined by

$$\begin{aligned} f\ (\{\}) &= \{\} \\ f\ (\{l:g\}) &= (l,g) \odot f(g) \\ f\ (g_1 \cup g_2) &= f(g_1) \cup f(g_2) \end{aligned}$$

where $\odot$ is a given binary operator. Different choices of $\odot$ define different recursive functions. For simplicity, the

definition above is abbreviated to

$$\textbf{sfun}\ \ f\ (\{l : g\}) = (l, g) \odot f(g).$$

Note that even for a graph g having cycles, the computation of $f(g)$ always terminates under the usual *recursive semantics* where all recursive calls are memorized and their results are reused to avoid entering infinite loops.

As a simple example, we may use the following recursive function $a2d\_xc$ to replace all edges labeled `a` by `d` and skip all edges labeled `c` for the graph in Figure 5(a).

$$\begin{aligned}\textbf{sfun}\ \ a2d\_xc\ (\{l : g\}) = {} & \textbf{if}\ l = \texttt{a}\ \textbf{then}\ \{\texttt{d} : a2d\_xc(g)\} \\ & \textbf{else if}\ l = \texttt{c}\ \textbf{then}\ a2d\_xc(g) \\ & \textbf{else}\ \{l : a2d\_xc(g)\}\end{aligned}$$

We naturally extend the structural recursion above so that it allows mutual recursion. Any mutually recursive functions can be merged into one by the standard tupling method [24].

C. Bidirectional Semantics of UnCAL

A query in UnCAL is usually run in forward direction: under an environment (a mapping from variables to graphs) ρ, a query Q generates a result graph denoted by $\mathcal{F}[\![Q]\!]\rho$.

Let $g = \mathcal{F}[\![Q]\!]\rho$ be a result graph. Assume that a user has edited it into g'. For example, one may add a new subgraph, modify some labels, and delete several edges. In our previous work [18], we gave a *backward semantics* that properly reflects back the editing to the original inputs. Formally speaking, given the modified result graph g' and the original input environments ρ, we presented a method which computes the modified environment $\rho' = \mathcal{B}[\![Q]\!](\rho, g')$.

By "properly reflecting back" (or *well-behaved*), we mean the following two properties to hold:

$$\frac{\mathcal{F}[\![Q]\!]\rho = g}{\mathcal{B}[\![Q]\!](\rho, g) = \rho}(\textsc{GetPut})$$

$$\frac{\mathcal{B}[\![Q]\!](\rho, g') = \rho'}{\mathcal{B}[\![Q]\!](\rho, \mathcal{F}[\![Q]\!]\rho') = \rho'}(\textsc{WPutGet})$$

The (GETPUT) property says that if no change is made on the output g, then there should occur no change on the input environment. The (WPUTGET) property is an unrestricted version of (PUTGET) property appeared in [14], which requires $g' \in \text{Range}(\mathcal{F}[\![Q]\!])$ and $\mathcal{B}[\![Q]\!](\rho, g') = \rho'$ to imply $\mathcal{F}[\![Q]\!]\rho' = g'$. The (PUTGET) property states that if a result graph is modified to g' which is in the range of the forward evaluation, then this modification can be reflected to the source such that a forward evaluation will produce the same result g'. In contrast, the (WPUTGET) property allows the modified result to be different from the result obtained by backward evaluation followed by forward evaluation, but require both to have the same effect on the original source if backward evaluation is applied again. This property enables us to make flexible modifications on the result graphs.

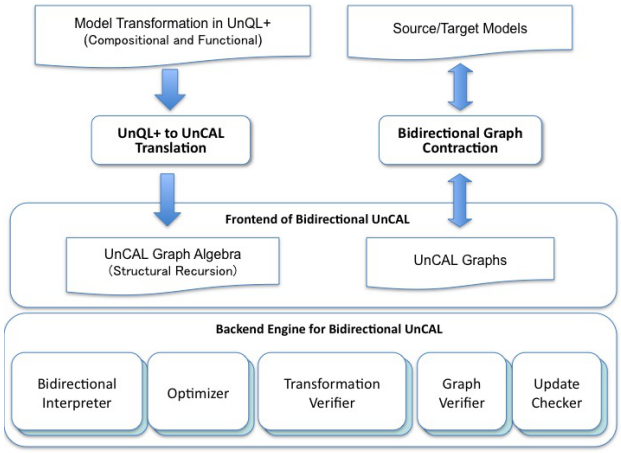


Figure 6. GRoundTram Implementation on Bidirectional UnCAL Engine

IV. Design and Implementation of GRoundTram

Let us now turn to show the technical details in design and implementation of the GRoundTram system, whose basic functions have been demonstrated in Section II.

Figure 6 depicts the architecture of the system. We provide a new user-friendly model transformation language UnQL$^+$ which is *functional* (rather than rule-based as in many existing tools) and *compositional* with high modularity for reuse and maintenance, and we accept 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 [18], a graph and graph transformation verifier [23], an optimizer to improve efficiency [25], and a checker of valid updates in the backward transformation [26]. 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.

In the rest of this section, we will focus on the exploration of UnQL$^+$, the bidirectional graph contraction algorithm, and the translation from UnQL$^+$ to UnCAL.

A. Model Transformation in UnQL$^+$

UnQL$^+$ is the language the GRoundTram system provides for users to describe (bidirectional) model transformations. It is an extension of the well-known UnQL [19], a graph querying language, which is compositional and can be implemented by FO (TC) (first order with transitive closure) with time complexity of PTIME for graph querying.

Figure 7 gives the core syntax of UnQL$^+$. A graph transformation is described by a template expression to construct a graph from graphs that are bound by graph

(template)	T	::=	$\{L:T,\ldots,L:T\} \mid T \cup T \mid \$g$
		|	**if** BC **then** T **else** T
		|	**select** T **where** $B,\ldots,B$
		|	**replace** $Rp \to \$G$ **by** T **in** T **where** $B,\ldots,B$
		|	**extend** $Rp \to \$G$ **with** T **in** T **where** $B,\ldots,B$
		|	**delete** $Rp \to \$G$ **in** T **where** $B,\ldots,B$
(binding)	B	::=	Gp **in** $\$G \mid BC$
(condition)	BC	::=	not $BC \mid BC$ and $BC \mid BC$ or BC
		|	isEmpty($\$G$) $\mid L = L \mid L \neq L \mid L < L \mid L \leq L$
(label)	L	::=	$\$l \mid$ a
(label pattern)	Lp	::=	$\$l \mid Rp$
(graph pattern)	Gp	::=	$\$G \mid \{Lp:Gp,\ldots,Lp:Gp\}$
(regular path pattern)	Rp	::=	a $\mid$ _ $\mid Rp.Rp \mid (Rp\|Rp) \mid Rp? \mid Rp* \mid Rp+$

Figure 7. Syntax of UnQL$^+$

variables. The expression $\{l_1 : t_1, \ldots, l_n : t_n\}$ creates a new node having n outgoing edges labeled l_i and pointing to the root of the graph computed from t_i. The union $g_1 \cup g_2$ constructs a graph with a root sharing the roots of g_1 and g_2. The variable expression $\$g$ returns the graph that are bound by $\$g$ in the environment (i.e., the mapping from variables to graphs). The conditional expression has the usual meaning, choosing different branch according to the (binding) condition B.

Like other query languages, UnQL$^+$ has a convenient template expression **select** t **where** bs, which is to select the subgraphs satisfying the condition sequence bs, bind them to variables, and construct a result according to the template expression t. For instance, the following query extracts all persistent classes from the class model in Figure 4, which is assumed to be bound by $\$db$.

$$\begin{array}{l}\textbf{select}\ \$class\ \textbf{where}\\ \quad \{Association.(src|dest).Class : \$class\}\ \textbf{in}\ \$db,\\ \quad \{is\_persistent : \{Boolean : true\}\}\ \textbf{in}\ \$class\end{array}$$

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 $\$db$, while the second condition is to ensure that the class is persistent.

In model transformation, one often wants to replace a subgraph satisfying certain condition by another graph, and it is onerous to describe these kinds of graph transformations using *select-where* because some context structure is required to be copied and propagated. To this end, we introduce three new template expressions, namely, *replace-where*, with later *extend-where* and *delete-where*.

- The *replace-where* expression is to replace a subgraph by a new graph. Consider the class model again, prefixing every name of the class by "class_" can be specified as follows. Note that "^" is a built-in function for string concatenation.

  replace _$*$.Class.name.string $\to \$u$
  by {("class_"^$\$name$) : {}} **in** $\$db$
  where {$\$name$: {}} **in** $\$u$

- The *delete-where* expression is used to describe the deletion of a part of the graph. For instance, we may eliminate all persistent classes by

  delete Association.(src|dest).Class $\to \$class$ **in** $\$db$
  where {is_persistent.Boolean : $true$} **in** $\$class$

  where the subgraph matched with $class will be deleted from its original graph $\$db$.
- The *extend-where* expression is to extend a graph with another graph. For example, we write the following transformation to add date information to each class.

  extend _$*$.class $\to \$c$ **with** {date : "2008/8/4"}
  in $\$db$

Unlike most rule-based model transformation languages where model transformation composition is not straightforwardly supported [12], UnQL$^+$ is functional and compositional; smaller model transformations can be composed to form a bigger one as demonstrated in Section II and will be seen in Section V.

B. Bidirectional Graph Contraction

As explained in Section III, our graph model is based on bisimulation equivalence, which means bisimilar graphs cannot be distinguished. Moreover, since UnCAL is based on bisimulation, transformation may introduce redundant nodes that are bisimilar to each other. Therefore, a normalization phase after transformation is required when such redundancy has to be eliminated.

Fortunately, it is known that for any set of graphs that are bisimilar with each other, there exists a unique normal form up to isomorphism and we can obtain the normal form after transformation using the partition refinement algorithm by Paige and Tarjan [27]. It's complexity is $O(|E| \log |V|)$ where $|E|$ and $|V|$ are the number of edges and nodes, respectively, and we consider that it is acceptable in practice.

Although this algorithm works on node-labeled graphs, we lift the algorithm to our edge-labeled graph model as described in [19]. After contraction, no pairs of nodes are bisimilar to each other. In particular, leaf nodes (nodes that have no outgoing edges) are bisimilar to each other, so they all shrink to one node.

We carefully design our contraction algorithm so that it forms a well-behaved bidirectional transformation that satisfies (GETPUT) and (WPUTGET) properties explained in Section III-C. For example, if an edge is inserted between a pair of contracted nodes, then the edge is uncontracted to multiple edges connecting corresponding bisimilar nodes. If the origin and the destination of the edge are both multiple, then only connections between pair of edges that was originally connected are established, although all-to-all connection also satisfies well-behavedness.

C. Translating UnQL$^+$ to UnCAL

UnQL$^+$ is different from UnCAL in that it uses four important template expressions, namely *select, replace, extend, delete*, to describe graph transformation rather than using structural recursion. In this section, we show that all these template expressions can be translated to structural recursions in UnCAL.

The *select* expression, which is inherited from UnQL, can be translated to structural expression in [19], whose explanation is omitted here. The *delete* and *extend* expressions can be defined in terms of the *replace* expression as:

$$\textbf{delete}\ \ Rp \to \$v\ \ \textbf{in}\ \ e_1\,\textbf{where}\ \ bs \\ \Rightarrow\ \ \textbf{replace}\ \ Rp \to \$v\ \ \textbf{by}\ \ \{\}\ \ \textbf{in}\ \ e_1\,\textbf{where}\ \ bs$$

$$\textbf{extend}\ \ Rp \to \$v\ \ \textbf{with}\ \ e_1\ \textbf{in}\ \ e_2\,\textbf{where}\ \ bs \\ \Rightarrow\ \ \textbf{replace}\ \ Rp \to \$v\ \ \textbf{by}\ \ \$v \cup e_1\ \textbf{in}\ \ e_2\,\textbf{where}\ \ bs$$

Therefore, what we need to show is how the *replace* expression is translated into structural recursion.

Our idea for this translation is to use structural recursion to simulate the behavior of deterministic finite automaton (DFA) for finding the nodes in the graph where the replace operation is to be applied. Now, consider the following general form of the *replace* expression.

$$\textbf{replace}\ \ Rp \to \$v\ \ \textbf{by}\ \ e_1\ \ \textbf{in}\ \ e_2\ \ \textbf{where}\ \ bs$$

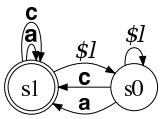
First, we translate the regular path pattern Rp into a DFA $(Q, \Sigma_{\$l}, \delta, q_0, F)$, where $Q = \{q_0, \ldots, q_N\}$ is a finite set of states, $\Sigma_{\$l} = \Sigma \cup \{\$l\}$ (where $\Sigma = \{l_0, \ldots, l_K\}$) is a finite set of labels used in Rp, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is a set of accept states. We use special label $\$l$ to denote a label other than labels used in Rp.

Then, we introduce $N+1$ functions $h_{q_0}, \ldots, h_{q_N}$, where function h_{q_i} corresponds to state q_i, and define each function h_{q_i} as a structural recursion in the following way. For each label $l \in \Sigma_{\$l}$, we define

$$h_{q_i}(\{l : \$v\}) \quad = \quad e_{ij}$$

$$\begin{array}{ll}\textbf{replace} & \_*.(\mathtt{a}|\mathtt{c}) -> \$v \\ \textbf{by} & \$v' \\ \textbf{in} & \$db \\ \textbf{where} & \{\mathtt{g} : \$u\}\ \ \textbf{in}\ \ \$v, \\ & \{\_*.d : \$v'\}\ \ \textbf{in}\ \ \$v\end{array}$$

(a) A replace expression



(b) DFA for $\_*.(\mathtt{a}|\mathtt{c})$

$$\begin{array}{ll}\textbf{let} & \textbf{sfun}\ h_{s0}(\{\mathtt{a} : \$v\}) = \textbf{if}\ \mathsf{isEmpty}(e_1)\ \textbf{then}\ \{\mathtt{a} : h_{s1}(\$v)\} \\ & \qquad\qquad\qquad\qquad \textbf{else}\ \{\mathtt{a} : e_2\} \\ & \quad | \quad h_{s0}(\{\mathtt{c} : \$v\}) = \textbf{if}\ \mathsf{isEmpty}(e_1)\ \textbf{then}\ \{\mathtt{c} : h_{s1}(\$v)\} \\ & \qquad\qquad\qquad\qquad \textbf{else}\ \{\mathtt{c} : e_2\} \\ & \quad | \quad h_{s0}(\{\$l : \$v\}) = \{\$l : h_{s0}(\$v)\} \\ & \textbf{sfun}\ h_{s1}(\{\mathtt{a} : \$v\}) = \textbf{if}\ \mathsf{isEmpty}(e_1)\ \textbf{then}\ \{\mathtt{a} : h_{s1}(\$v)\} \\ & \qquad\qquad\qquad\qquad \textbf{else}\ \{\mathtt{a} : e_2\} \\ & \quad | \quad h_{s1}(\{\mathtt{c} : \$v\}) = \textbf{if}\ \mathsf{isEmpty}(e_1)\ \textbf{then}\ \{\mathtt{c} : h_{s1}(\$v)\} \\ & \qquad\qquad\qquad\qquad \textbf{else}\ \{\mathtt{c} : e_2\} \\ & \quad | \quad h_{s1}(\{\$l : \$v\}) = \{\$l : h_{s0}(\$v)\} \\ \textbf{in} & h_{s0}(\$db) \\ \textbf{where} & e_1 \equiv \textbf{select}\ \ \{\text{“}\mathtt{found}\text{”} : \{\}\} \\ & \qquad\quad \textbf{where}\ (\{\mathtt{g} : \$u\}\ \textbf{in}\ \$v), (\{\_*.d : \$v'\}\ \textbf{in}\ \$v) \\ & e_2 \equiv \textbf{select}\ \ \$v' \\ & \qquad\quad \textbf{where}\ (\{\mathtt{g} : \$u\}\ \textbf{in}\ \$v), (\{\_*.d : \$v'\}\ \textbf{in}\ \$v)\end{array}$$

(c) Translated structural recursion

Figure 8. A Translation Example

and construct a graph with expression e_{ij} by considering two cases. If $\delta(q_i, l) \notin F$ (i.e., transition from state q_i through label l does not reach to an accept state), then we keep the context by propagating l and continue the recursive computation by defining

$$e_{ij} = \{l : h_{\delta(q_i,l)}(\$v)\}.$$

Otherwise we check whether $\$v$ satisfies condition bs, and if it is, we replace the graph with the query result of e_1 satisfying bs:

$$\begin{array}{rcl} e_{ij} & = & \textbf{if}\ \mathsf{isEmpty}\,(\textbf{select}\ \ \{\text{“}\mathtt{found}\text{”}\}\ \ \textbf{where}\ \ bs) \\ & & \textbf{then}\ \{l : h_{\delta(q_i,l)}(\$v)\} \\ & & \textbf{else}\ \{l : (\textbf{select}\ \ e_1\ \ \textbf{where}\ \ bs)\}. \end{array}$$

The condition $\mathsf{isEmpty}(...)$ in the **if**-expression checks whether the condition bs holds. Note that since e_1 might be evaluated to $\{\}$, the checking expression should not be $\textbf{select}\, e_1\, \textbf{where}\ \ bs$.

Example 1. Our algorithm maps a *replace* expression shown in Figure 8(a) to the structural recursion in Figure 8(c) via the DFA obtained from $\_*.(\mathtt{a}|\mathtt{c})$ in Figure 8(b).

V. EVALUATION AND APPLICATIONS

In this section, we demonstrate the power (expressiveness and efficiency) of the GRoundTram system through development of a known nontrivial (bidirectional) model transformation between UML class diagrams and relational databases, and highlight its usefulness in practice by giving a list of important applications developed by other groups using the GRoundTram system.

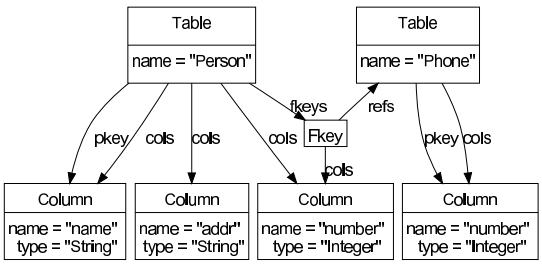


Figure 9. An RDB Model

A. Developing Bidirectional Class2RDB

Class2RDB is a known model transformation, which was proposed at [28] as a common benchmark example to all the participants of the workshop for comparing and contrasting various kinds of approaches to model transformations. Class2RDB maps Class models to RDB models. For instance, it transforms the Class model in Figure 3 into an RDB model in Figure 9. Simply speaking, Class2RDB maps each persistent class in a Class model to a table in a RDB model. All attributes of the class or its subclasses are mapped to columns in the corresponding table. If a primary attribute belongs to the class, a `pkey` pointing from the table to the corresponding column is established. If an attribute belongs to its subclass which is persistent, a foreign key to the corresponding table is established.

We show that UnQL$^+$ is powerful enough to (compositionally) describe the forward transformation (from class diagrams to relational databases) while achieving the backward transformation for free in our framework. Figure 10 gives the whole transformation in UnQL$^+$. Let us briefly explain how this UnQL$^+$ program is developed by splitting the transformation into two steps. In the first step, every persistent class is mapped to a table which is connected with its columns according to attributes of the class and its subclasses. All subclasses are collected by regular path patterns as shown in Section IV-A. If necessary, references `pkey` and `fkeys` are added by an **extend** construct in UnQL$^+$, provided that references `refs` of `Fkey` do not point to the referring table because the table may not have been constructed yet. They point to the name of the referring table instead. In the second step, each name pointed by `refs` is replaced by the corresponding table by using a **replace** construct.

B. Optimization and Efficiency

Next, we show that both forward and backward transformations can be run efficiently in a scalable manner, while the inefficiency due to composition can be automatically removed though our fusion optimization.

Table I summarizes performance of bidirectional transformations on various compositional transformations, running on MacOSX over MacBookPro 17 inch, with 3.06 GHz Intel Core 2 Duo CPU. Algorithm for edge-renaming is used for the measurement, and no modification is actually given, since the presence of modification does not significantly affect the running time. Last column shows running time with rewriting optimization applied. *PIM2PSM* stands for Platform Independent Model to Platform Specific Model transformation, *C2Osel* for transformation of customer oriented database into order oriented database, followed by a simple selection, and *UnQL* for the example that is extracted from our previous paper [29], which was borrowed from [19]. It is a composition of two structural recursions. The numbers in parentheses show how often the fusion transformation happened. Our rewriting led to performance improvements in both directions. As the run-time optimization, unreachable parts are removed after every application of UnCAL structural recursion operator. For *C2Osel'*, this optimization is turned off. This run-time optimization is effective when each composed transformation has high selectivity (generates small output from large input) while fusion is effective when the selectivity is low. Slowdown in *C2Osel* after rewriting accounts for this trade-off. For the principles of this rewriting optimization, please refer to our separate paper [25]. You can test other optimization like subgraph computation optimization in our project website [20]. Figure 11 shows how the size of the source model affects time to execute $a2d\_xc$ transformation introduced in Section III, in both directions. Lattice-like regularly shaped strongly-connected graphs are used as the source. These execution

Table I
SUMMARY OF EXPERIMENTS (RUNNING TIME IS IN CPU SECONDS)

	direction	no rewriting	rewriting	
Class2RDB	forward	1.18	0.68	
	backward	14.5	7.90	
PIM2PSM	forward	0.08	0.07	(13)
	backward	1.62	0.75	
C2Osel	forward	0.04	0.05	(11)
	backward	0.26	0.27	
C2Osel'	forward	0.05	0.04	(11)
	backward	2.56	1.27	
UnQL	forward	0.036	0.007	(1)
	backward	0.83	0.69	

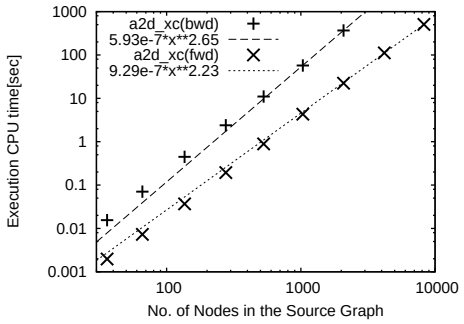


Figure 11. Transformation Time v.s. Source Graph Size

```
select $tables_step2 where
  $tables_step1 in
    (select $tables where
       {Class:$class} in (select $assoc where {Association.(src|dest):$assoc} in $db),
       {is_persistent.Boolean:true} in $class,
       $dests in (select {Class:$dest} where {(src_of.Association.dest.Class)+:$dest} in $class),
       $related in ({Class:$class} U $dests),
       $cols in (select {cols:{Column:{name:$n,type:$t}}} where {Class.attrs.Attribute:{name:$n,type:$t}} in $related),
       $tables in (select {Table:{name:$cname} U $cols} where {name:$cname} in $class),
       $tables in (extend Table -> $table with $pkeys U $fkeys in $tables where
                     {cols:$cols} in $table,
                     {Column.name.String:{$cname:{}}} in $cols,
                     $pkeys in (select {pkey:$cols} where
                                  {attrs.Attribute: {is_primary.Boolean:true, name.String:{$pname:{}}}} in $class,
                                  $cname = $pname),
                     $fkeys in (select {fkeys:{Fkey:{cols:$cols, ref:$ref}}} where
                                  {Class:{is_persistent.Boolean:true,
                                          attrs.Attribute.name.String:{$aname:{}}, name:$ref}} in $dests,
                                  $cname = $aname))),
  $tables_step2 in (replace Table.fkeys.Fkey.ref -> $ref by {Table:$table} in $tables where
                     {Table:$table} in $tables_step1,
                     {String:{$rname:{}}} in $ref,
                     {name.String:{$tname:{}}} in $table,
                     $tname = $rname)
```

Figure 10. Class2RDB in UnQL$^+$

times match the complexity of PTIME that is mentioned in Section IV-A, up to relatively large size (several thousands of nodes) of graphs.

C. Other Applications

The GRoundTram website [20] provides a bunch of examples, big and small, and all the examples presented in this paper can be tried through the demo website. In addition, we would like to give a rough idea about the status of current uses of the GRoundTram system by listing applications that have been or are being developed with GRoundTram by other groups: Bidirectional Feature Model Transformation (Peking University), Bidirectional Transformation between VDM Specification and Java Implementation (another group in National Institute of Informatics), Bidirectional Transformation between Simulink Diagrams and UML Diagrams (Waseda University), Bidirectionalizing ATL with GRoundTram (Shibaura Institute of Technology), and Bidirectional Refactoring of Java Codes (Open University & Shanghai Jiao Tong University). All these indicate the promise of GRoundTram in practice.

VI. Related Work

Besides the related work in the introduction, we highlight some others related to graph-based model transformation and linguistic approach to bidirectional transformation.

Our work is much related to research on model transformation based on graph transformation. AGG [30] is a well-known rule-based visual tool that supports typed (attributed) graph transformations including type inheritance and multiplicities. Triple Graph Grammars (TGG) [7,31] aims at the declarative specification of model to model integration rules. Different from these rule-based ones, our approach is functional in which model transformation composition for systematic development and its automatic optimization are supported. As far as we are aware, this is the first nontrivial *functional and algebraic framework for model transformation*.

This work has been inspired by recent work on linguistic approach to bidirectionalization of tree transformation [14–17] for tree data synchronization. One important feature of these systems is a clear bidirectional semantics, which does not exist in most existing bidirectional model transformation systems [2]. Although some attempts have been made [5,6], it remains as a challenge to provide a general bidirectional framework for graphs which are more complicated than trees, and this work is a big step to this direction.

This work grows out of our two-year effort in realizing the emerging idea presented in a short paper [32]. The UnQL$^+$ is based on the graph query language UnQL [19] but it is significantly extended with a powerful language construct *replace* that can handle transformation context. It is also worth noting that a simple *replace* expression was studied in [29] but it can neither deal with regular path expressions nor treat multiple graph databases.

VII. Conclusions

In this paper, we propose a novel algebraic framework to support systematic development of bidirectional model transformation. Different from many existing frameworks that are rule-based, our framework is functional and algebraic, which is based on a graph algebra and structural recursion. Our new framework supports systematic development of model transformations in a compositional manner, has a clear semantics for bidirectional model transformation, and can be efficiently implemented.

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] 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)*. ACM Press, Nov. 2007, pp. 164–173.

[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*. Springer Verlag, 2006, pp. 543–557.

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

[9] R. Lämmel, "Coupled Software Transformations (Extended Abstract)," in *First International Workshop on Software Evolution Transformations*, Nov. 2004.

[10] J. Grundy, J. Hosking, and W. B. Mugridge, "Inconsistency management for multiple-view software development environments," *IEEE Trans. Softw. Eng.*, vol. 24, no. 11, pp. 960–981, 1998.

[11] M. Garcia, "Bidirectional synchronization of multiple views of software models," in *Proceedings of DSML-2008*, ser. CEUR-WS, vol. 324, 2008, pp. 7–19.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraph - static and dynamic graph drawing tools," in *GRAPH DRAWING SOFTWARE*. Springer-Verlag, 2003, pp. 127–148.

[22] 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.

[23] 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.

[24] Z. Hu, H. Iwasaki, M. Takeichi, and A. Takano, "Tupling calculation eliminates multiple data traversals," in *ACM SIGPLAN International Conference on Functional Programming (ICFP'97)*. Amsterdam, The Netherlands: ACM Press, Jun. 1997, pp. 164–175.

[25] 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.

[26] 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.

[27] R. Paige and R. Tarjan, "Three partition refinement algorithms," *SIAM Journal of Computing*, vol. 16, no. 6, pp. 973–988, 1987.

[28] J. Bezivin, B. Rumpe, A. Schürr, and L. Tratt, "Model transformation in practice workshop announcement," in *Satellite Events at the MoDELS 2005 Conference*. Springer-Verlag, 2005, pp. 120–127.

[29] S. Hidaka, Z. Hu, H. Kato, and K. Nakano, "Towards a compositional approach to model transformation for software development," in *SAC'09: Proceedings of the 2009 ACM symposium on Applied Computing*. New York, NY, USA: ACM, 2009, pp. 468–475.

[30] G. Taentzer, "AGG: A graph transformation environment for modeling and validation of software," in *AGTIVE*, ser. LNCS, J. L. Pfaltz, M. Nagl, and B. Böhlen, Eds., vol. 3062. Springer, 2003, pp. 446–453.

[31] A. Königs and A. Schürr, "Tool integration with triple graph grammars - a survey," *Electronic Notes in Theoretical Computer Science*, vol. 148, no. 1, pp. 113–150, February 2006.

[32] S. Hidaka, Z. Hu, H. Kato, and K. Nakano, "A compositional approach to bidirectional model transformation," in *ICSE Companion*. IEEE, 2009, pp. 235–238.