From State- to Delta-based Bidirectional Model Transformations: the Symmetric Case

Zinovy Diskin[1], Yingfei Xiong[1], Krzysztof Czarnecki[1], Hartmut Ehrig[2], Frank Hermann[2,3], and Fernando Orejas[4]

[1] Generative Software Development Lab, University of Waterloo, Canada
{zdiskin,yingfei,kczarnec}@gsd.uwaterloo.ca
[2] Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, Germany, ehrig@cs.tu-berlin.de
[3] Interdisciplinary Center for Security, Reliability and Trust,
Université du Luxembourg, Frank.Hermann@uni.lu
[4] Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Barcelona, Spain, orejas@lsi.upc.edu

Abstract. A bidirectional transformation (BX) keeps a pair of interrelated models synchronized. Symmetric BXs are those for which neither model in the pair fully determines the other. We build two algebraic frameworks for symmetric BXs, with one correctly implementing the other, and both being delta-based generalizations of known state-based frameworks. We identify two new algebraic laws—weak undoability and weak invertibility, which capture important semantics of BX and are useful for both state- and delta-based settings. Our approach also provides a flexible tool architecture adaptable to different user's needs.

1 Introduction

Keeping a system of models mutually consistent (model synchronization) is vital for model-driven engineering. In a typical scenario, given a pair of inter-related models, changes in either of them are to be propagated to the other to restore consistency. This setting is often referred to as bidirectional model transformation (BX) [3].

As noted by Stevens [15], despite early availability of several BX tools on the market, they did not gain much user appreciation because of semantic issues. Indeed, to avoid surprises, a user should clearly understand the behavior of synchronization procedures implemented by the tool. To formalize the semantics of BX tools and guide their implementation, algebraic frameworks for BX have been studied intensively [8, 15, 6, 19, 12].

The majority of algebraic BX frameworks (including all those cited above) are *state-based*. Synchronizing operations take the states of models before and after update as input, and produce new states of models as output. This design assumes that model alignment, i.e., discovering relations (*deltas*) between models, is done by update propagating procedures themselves. Hence, two quite different operations—heuristics-based delta discovery and algebraic delta propagation—are merged, which causes several theoretical and practical problems [2, 5]; we will discuss them in Section 2.2 after considering several basic examples.

To separate delta discovery and propagation, several researchers proposed to build *delta-based* frameworks [4, 2, 5, 11], in which propagation operations use deltas as input and output rather than compute them internally. Such frameworks (a general one [5] and a tree-oriented [2]) have been built for the *asymmetric* BX case, in which one model in the pair is a view of the other and hence does not contain any new information. In practice, however, it is often the case that two models share some information but each of them contains something new not present in the other; following [11], we call this case *symmetric* BX. The symmetric case has been considered in the state-based setting [13, 15, 6, 11], yet a precise delta-based symmetric framework has been an open issue.

In this paper, we fill the gap and develop a delta-based framework for symmetric BX. We build two algebraic structures, *symmetric delta lenses* and *(consistency) maintainers*, which comprise delta-based synchronization operations and laws they must satisfy. Lenses are more abstract and specify an interface of a model synchronization tool; maintainers are closer to implementation and allow the tool to reuse an infrastructure for delta composition. We show that 1) a lens can be built from a maintainer, and 2) the lens's laws are derived from the maintainer's laws so that a desirable lens's behavior is guaranteed when the lens is implemented by a suitable maintainer.

The second major contribution of the paper is the introduction of two new algebraic laws: weak invertibility and weak undoability. A long-standing problem in existing symmetric BX frameworks is that the basic laws (correctness and Hippocraticness [13, 15]) are not enough to ensure reasonable BX behavior, whereas more advanced laws like undoability [15] and invertibility [6] are known to be too strong and exclude many quite practical BXs. Our new laws solve this problem by reshaping strong laws into a weaker form that allows for reasonable symmetric BXs and yet prohibits BXs with unwanted behavior.

The paper is organized as follows. Section 2 analyzes an example and identifies three problems of state-based BXs that motivate our work on delta-based BXs. We present sd-lenses in Section 3 and maintainers in Section 4. Section 5 discusses related work, and Section 6 concludes the paper. Proofs and examples omitted in the paper can be found in its longer version [7].

2 The need for deltas

We begin with an example showing how state-based frameworks work and what their problems are. Then we explain why delta-based frameworks are needed.

2.1 Example

Figure 1 presents two related models A and B. The former specifies a class of Persons with their names and birth years, and the latter specifies Employees

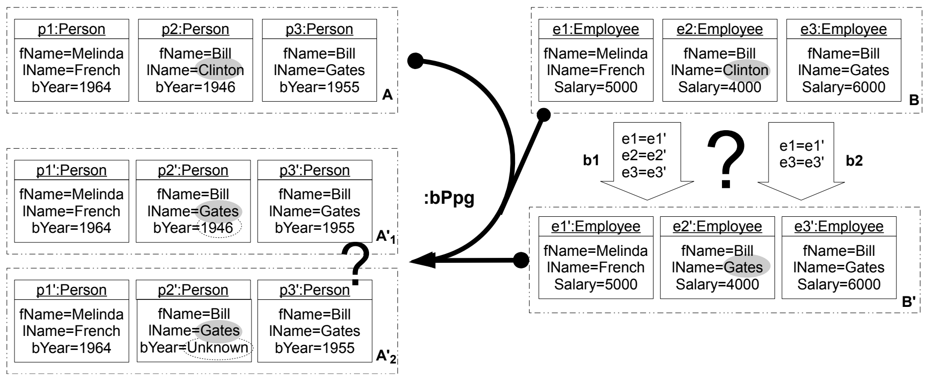


Fig. 1: The need of vertical deltas (updates)

with their names and salaries. Two models are considered consistent if the correspondence between Persons and Employees, inferred from the equality of their full names, is bijective. Initially models A and B are consistent, but then B is modified into B' and we need to propagate the change to the A side.

A suitable state-based BX framework designed for this task is trigonal systems [6]. Changes between the two sides are propagated by two ternary operations: forward propagation fPpg and backward propagation bPpg. When model B changes to B', operation bPpg takes the updated model B' and the original models B, A, and produces an updated model $A' = \mathsf{bPpg}(B', B, A)$. Forward propagation fPpg works similarly: $B' = \mathsf{fPpg}(A', A, B)$.

Figure 1 shows that two reasonable interpretations of the updated model B' are possible. Object $e2'$ may be understood as either a renamed version of $e2$, or a new object inserted into the model while $e2$ is deleted. The difference can be formally captured by specifying sets of pairs $(e, e') \in B{\times}B'$ with e and e' considered to represent the same object; we call this set $\simeq_v \subset B{\times}B'$ a *(vertical) sameness* relation. A triple $b = (B, \simeq_v, B')$ is called an *update delta* from B to B' and we write $b\colon B \to B'$. From $\simeq_v$ we can infer which objects were deleted, inserted, or modified. For example, $e2$ is deleted by delta $b2$ because it is not included in $b2$, but it is modified by $b1$ because it is declared to be the same as $e2'$ and the last names in $e2$ and $e2'$ are different.

Now we observe that two different deltas, $b1$ and $b2$, lead to two different synchronization results. To see that, we first define a correspondence between models A and B via full names of objects, i.e., we set a *(horizontal) sameness* relation $\simeq_h \subset A{\times}B$ between models A and B; in our case, it consists of three pairs (pi, ei), $i = 1, 2, 3$. Propagating delta $b1$ to the A side results in model A'_1: as objects $p2$ and $e2$, $e2$ and $e2'$ are the same, we merely apply modification of $e2$ to $p2$. However, propagation of delta $b2$ leads to model A'_2, which differs from A'_1 in the value of bYear: as object $e2$ is deleted and $e2'$ is inserted, object $p2$ is deleted and A-counterpart of $e2'$ — a new object $p2'$ — is inserted, but its birth date is unknown. Thus, propagation essentially depends on deltas, and

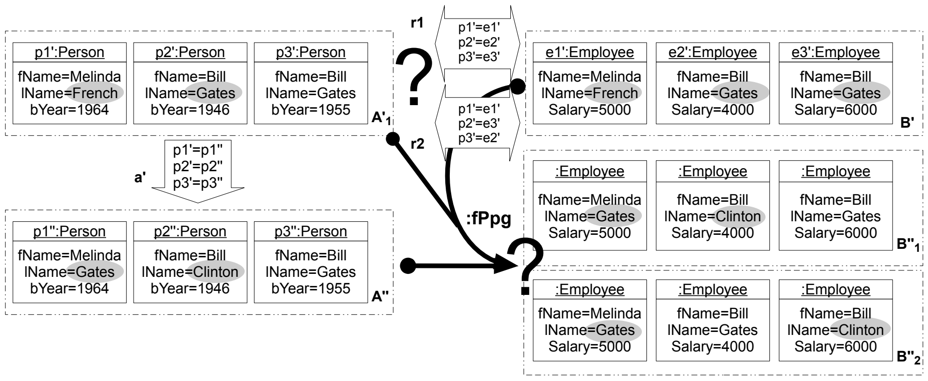


Fig. 2: The need of horizontal delta (correspondence)

propagation operation bPpg has to compute them using some heuristics, and then propagate the change.

To unify terminology and notation, we call a triple $r = (A, \simeq_h, B)$ a *correspondence* or *horizontal* delta from A to B and write $r\colon A \leftrightarrow B$; update deltas are *vertical*. Importantly, the same models A and B may have different correspondence deltas between them. For example, suppose that a user reviews the updated model $A1'$ and discovers that the change is mistaken: it is Melinda French who gets married and changes her last name, but not Bill Clinton. Then the user changes names of objects $p1$ and $p2$ to, respectively, Melinda Gates and Bill Clinton, as shown in Fig. 2 with update delta $a'\colon A'_1 \to A''$. To propagate the update to the B-side, we need to relate models $A1'$ and B' and rename the corresponding Employees. However, because there are two "Bill Gates" in both models, two cases of correspondences, $r1$ and $r2$ in Fig. 2, are possible, which lead to two different results: B''_1 and B''_2. Of course, from the previous propagation we know that the correct delta is $r1$, but since this delta does not explicitly occur in the output of operation bPpg, forward propagation fPpg does not know it and has to infer it from the current states of the models.

2.2 Unweaving delta discovery and propagation

$$\begin{tikzcd}
\boxed{A} \arrow[rr, leftrightarrow, "r"] \arrow[d, dashed, "a1"'] & & \boxed{B} \arrow[d, "b1"] \\
A'_1 \arrow[rr, dashed, leftrightarrow, "r1"] \arrow[d, "a'"'] & \Swarrow\text{:bPpg} & \boxed{B'_1} \arrow[d, dashed, "b'"] \\
\boxed{A''} \arrow[rr, dashed, leftrightarrow, "r''"] & \Searrow\text{:fPpg} & B''
\end{tikzcd}$$

Problems of merging delta discovery into update propagation. First, such a merge, as presented in state-based frameworks, essentially complicates propagation operations and their semantics. Delta discovery is an independent operation with its own laws [1, 16], and is usually far more complex than propagation as such. Weaving delta discovery into update propagation complicates the laws of the latter and makes its behavior less predictable.

Second, it unnecessarily complicates support of update sequences. Indeed, our example can be specified as shown by the inset diagram above (input nodes are framed and input arrows are solid; output elements are, respectively, non-framed and dashed). It shows that the output horizontal delta $r1$ produced by bPpg must be the input delta for fPpg. However, in a straightforward state-based implementation, operation fPpg computes the delta afresh, which may result in a different delta $r_1' \neq r_1$.

Third, our previous work [5] shows that similar problems appear in sequential composition of BX (think of another BX from B- to C-models) if vertical deltas are replaced by pairs of models, as is done in the state-based frameworks.

A solution to these three problems is to encapsulate delta propagation in a special module, which takes the horizontal and vertical deltas as input, and produces new vertical and horizontal deltas as shown in the inset diagram above; we call such a module a delta-based BX. It has a simple algebraic semantics, prevents erroneous composition of updates and BXs, and allows reusing deltas.

Implementation of deltas. Normally, only small parts of big models are updated, and implementing vertical deltas as sameness relations is very non-economic. A practical solution is to implement them operationally as edit sequences or as overriding deltas [18, 5]. Horizontal deltas can be seen as traceability links, which are maintained by many transformation tools. For either representation, deltas can be abstracted as arrows relating two models.

Managing deltas and tool architecture. Having a separate delta-propagating module provides a flexible tool architecture. For example, the state-based framework can be simulated if deltas are first discovered by a model differencing tool and then passed to the propagation module. If the two models are related by a transformation, horizontal deltas can be inferred from it — this architecture is used in SyncATL [17]. Hybrid interfaces (state-based for one dimension and delta-based for the other) are also possible, e.g., two incremental synchronization tools, based on TGG [9] and QVT [14], take vertical deltas as input and store horizontal deltas internally. An additional advantage of separating delta discovery from propagation is that the user may control the result of differencing and correct it if needed. Finally, if the synchronizer can be tightly coupled with the application, deltas can be obtained by recording the user operations within the applications; in this case, model differencing phase is not needed.

Although the tools mentioned above actually use a separated delta propagation module, they lack a precise specification of both their architecture and semantics of propagation procedures they guarantee. Filling the gap needs a precise definition of delta-based symmetric BX and a formal algebraic theory of delta propagation. Developing both of them is our goal for the rest of the paper.

3 Symmetric delta lenses

We first specify an algebraic structure modeling the very basic properties of update propagation (Section 3.1). Then we enrich the structure with more advanced laws of undoability and invertibility (Section 3.2).

3.1 The basic structure

We begin by defining the space of models and their vertical deltas as a graph with an additional structure representing do-nothing updates and update inversion; this structure makes the graph *reflexive* and *involutive*.

Definition 1 (Model space) A *model space* $\mathbf{A}$ is a graph $(\mathbf{M_A}, \mathbf{\Delta_A}, \$_\mathbf{A})$, whose nodes $A \in \mathbf{M_A}$ are called $\mathbf{A}$-*models*, arrows $a \in \mathbf{\Delta_A}$ are $\mathbf{A}$-model *deltas*, and $\$_\mathbf{A}$ is a quadruple of total unary "bookkeeping" functions $(\Box_{\mathbf{A}}\_\,, \_\Box_{\mathbf{A}}, \mathsf{id}_{\mathbf{A}}\_\,, \_^{\smile\mathbf{A}})$ (with "$\_$" being the placeholder) providing $\mathbf{A}$ with the structure of reflexive involutive graph explained below.

Functions $\Box_{\mathbf{A}}\_, \_\Box_{\mathbf{A}} : \mathbf{\Delta_A} \to \mathbf{M_A}$ provide deltas with their *source* and *target* models resp., and we write $a\colon A \to A'$ if $\Box_{\mathbf{A}} a = A$ and $a\Box_{\mathbf{A}} = A'$. Intuitively, we understand a as a delta resulting from some update to model A, i.e., as a triple $(A, \simeq_v, A')$ like those considered in Section 2.1. By an abuse of terminology, we will often call delta a an update from A to A' (though different sequences of update operations can result in the same delta).

Function $\mathsf{id}_{\mathbf{A}}\colon \mathbf{A} \to \mathbf{\Delta_A}$ assign to every model A a special *identity* delta $\mathsf{id}_{\mathbf{A}}A\colon A \to A$ that identically relates A to itself. Such a delta may be thought of as (the result of) an *idle* update to A, which does nothing. To capture this intuition formally, we need to introduce sequential composition of deltas and require $\mathsf{id}_{\mathbf{A}}$ to be its neutral unit (see [5] for details), but in this paper we do not consider vertical delta composition. However, we will later capture idleness of $\mathsf{id}_{\mathbf{A}}$-arrows wrt. their composition with horizontal deltas.

Finally, $\_^{\smile\mathbf{A}}$ is an unary operation of *delta inversion*: for $a\colon A \to A'$, arrow $a^{\smile\mathbf{A}}\colon A' \to A$ is the same delta traversed in the opposite direction. For example, the inverse of delta $a = (A, \simeq, A')\colon A \to A'$ in Fig. 2 with $\simeq\, = \{(p1, p1'), (p2, p2'), (p3, p3')\}$ is delta $a^{\smile} = (A', \simeq^{-1}, A)\colon A' \to A$ with $\simeq^{-1}\, = \{(p1', p1), (p2', p2), (p3', p3)\}$. It can be understood as the delta resulting from undoing update a: changing lNames of $p1'$ and $p2'$ to French and Gates resp.

The following evident laws are required (subscript $\mathbf{A}$ near $\_^{\smile}$ is omitted):

$$(\mathsf{id}_{\mathbf{A}}A)^{\smile} = \mathsf{id}_{\mathbf{A}}A \text{ for all } A \in \mathbf{M_A} \text{ and } (a^{\smile})^{\smile} = a \text{ for all } a \in \mathbf{\Delta_A},$$

which make operation $\_^{\smile}$ an *involution* and the graph *involutive*.

Thus, a model space is a reflexive involutive graph.

Now we introduce horizontal deltas as arrows between models in two model spaces, and come to the notion of *triple spaces*.

Definition 2 (Triple space) A *triple space* $\mathbf{R}\colon \mathbf{A} \leftrightarrow \mathbf{B}$ or $\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$ consists of a pair of models spaces $(\mathbf{A}, \mathbf{B})$, and a set $\mathbf{R}$ of arrows from $\mathbf{A}$-nodes to $\mathbf{B}$-nodes called *correspondence relations*, or just *corrs*. Formally, $\mathbf{R} = (\mathbf{M_A}, \mathbf{M_B}, \mathbf{\Delta_{AB}}, \$_{\mathbf{AB}})$ is a graph with $\mathbf{M_A} \cup \mathbf{M_B}$ being the set of nodes, $\mathbf{\Delta_{AB}}$ the set of arrows (corrs), and $\$_{\mathbf{AB}}$ consists of two functions, $\Box_{\mathbf{AB}}\_\colon \mathbf{\Delta_{AB}} \to \mathbf{M_A}$ and $\_\Box_{\mathbf{AB}} : \mathbf{\Delta_{AB}} \to \mathbf{M_B}$, providing corrs with their *source* and *target* models. For $r \in \mathbf{\Delta_{AB}}$, we write $r\colon A \leftrightarrow B$ if $\Box_{\mathbf{AB}} r = A$ and $r\Box_{\mathbf{AB}} = B$.

$$\begin{tikzcd}
\boxed{A} \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, phantom, "\Downarrow{:}\mathsf{fPpg}"] & \boxed{B} \arrow[d, dashed, "b"] \\
\boxed{A'} \arrow[r, leftrightarrow, dashed, "r'"] & B'
\end{tikzcd}
\qquad
\begin{tikzcd}
\boxed{A} \arrow[r, leftrightarrow, "r"] \arrow[d, dashed, "a"] \arrow[dr, phantom, "\Downarrow{:}\mathsf{bPpg}"] & \boxed{B} \arrow[d, "b"] \\
A' \arrow[r, leftrightarrow, dashed, "r'"] & \boxed{B'}
\end{tikzcd}
\qquad
\begin{tikzcd}
\boxed{A} \arrow[r, leftrightarrow, "r"] \arrow[d, "\mathsf{id}A"'] \arrow[dr, phantom, "\Downarrow{:}\mathsf{fPpg}"] & \boxed{B} \arrow[d, dashed, "\mathsf{id}B"] \\
\boxed{A} \arrow[r, leftrightarrow, dashed, "r"] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
\boxed{A} \arrow[r, leftrightarrow, "r"] \arrow[d, dashed, "\mathsf{id}A"'] \arrow[dr, phantom, "\Downarrow{:}\mathsf{bPpg}"] & \boxed{B} \arrow[d, "\mathsf{id}B"] \\
A \arrow[r, leftrightarrow, dashed, "r"] & \boxed{B}
\end{tikzcd}$$

(a) fPpg (b) bPpg (c) (IdPpg) law

Fig. 3: Stable sd-lens: operations (a,b) and the law (c)

To ease terminology, we will use term 'delta' generically for both updates (*vertical* deltas) and correspondences (*horizontal* deltas). We will also write bookkeeping functions, i.e., components of $\$_{\mathbf{A}}$, $\$_{\mathbf{B}}$, and $\$_{\mathbf{AB}}$ without subscripts.

Now we define operations modeling update propagation.

Definition 3 (sd-lenses) A *symmetric delta lens (sd-lens)* over a triple space $\mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$ is a pair of *forward* and *backward propagation* operations (note that backward propagation arrow goes from right to left)

$\mathsf{fPpg}\colon \mathbf{\Delta_A} \,{}^{\square}\!\!\times\, \mathbf{\Delta_{AB}} \to \mathbf{\Delta_B} \times_{\square} \mathbf{\Delta_{AB}}$ and $\mathsf{bPpg}\colon \mathbf{\Delta_A} \,{}_{\square}\!\!\times\, \mathbf{\Delta_{AB}} \leftarrow \mathbf{\Delta_B} \times^{\square} \mathbf{\Delta_{AB}}$

of arities shown in Fig. 3(a,b): input nodes are framed, input arrows are solid, and the output elements are non-framed and dashed. Figure 4 shows an example: operation fPpg takes deltas a and r and produces deltas b and r'.

Symbol ${}^{\square}\!\!\times$ in the formulas above denotes the subset of the respective Cartesian product consisting of all pairs of arrows with the same source: $\mathbf{\Delta_A} \,{}^{\square}\!\!\times\, \mathbf{\Delta_{AB}} = \{(a,r) \in \mathbf{\Delta_A} \times \mathbf{\Delta_{AB}} \colon\ \square_{\mathbf{A}} a = \square_{\mathbf{AB}} r\}$, and respectively $\mathbf{\Delta_B} \times_{\square} \mathbf{\Delta_{AB}} = \{(b,r) \in \mathbf{\Delta_B} \times \mathbf{\Delta_{AB}} \colon\ b\square_{\mathbf{B}} = r\square_{\mathbf{AB}}\}$ is the subset of pairs with the same target. Similarly, the meaning of symbols $\times^{\square}$ and ${}_{\square}\!\!\times$ is defined by diagram Fig. 3(b). We must also require right correspondence of the input and output pairs: for fPpg, if $(b,r') = \mathsf{fPpg}(a,r)$, then $\square b = r\square$ and $\square r' = a\square$, and for bPpg, if $(a,r') = \mathsf{bPpg}(b,r)$, then $\square a = \square r$ and $r'\square = b\square$. We call these and similar equations specifying relationships between arrows *incidence conditions*.

Note that the arity diagrams unambiguously specify all required incidence conditions, and their explicit string-based formulation as above can be omitted. In fact, operations like fPpg and bPpg act upon arrow diagrams, and can be accurately formalized in terms of *diagram algebra* [4], which allows one to avoid bulky formulation of incidence conditions. Below we will use the arity diagram of an operation as a part of the definition and write $\boxtimes$ for ${}^{\square}\!\!\times$, $\times_{\square}$, $\times^{\square}$, or ${}_{\square}\!\!\times$.

The small double arrows in the middle labeled by :fPpg, :bPpg indicate that the squares are *application instances* of the operations (other instances are are formed by other arguments). In the same manner we could write also $a{:}\mathbf{\Delta_A}$, $r{:}\mathbf{\Delta_{AB}}$ *etc*, but we omit these to avoid too heavy notation.

It is convenient to use also the following notation: for the situation in Fig. 3(a), we write $a.\mathsf{fPpg}(r)$ for b and $r.\mathsf{fPpg}(a)$ for r', and similarly for bPpg. To resolve ambiguity, we always use a, b to denote deltas in $\mathbf{A}, \mathbf{B}$, and r to denote correspondences.

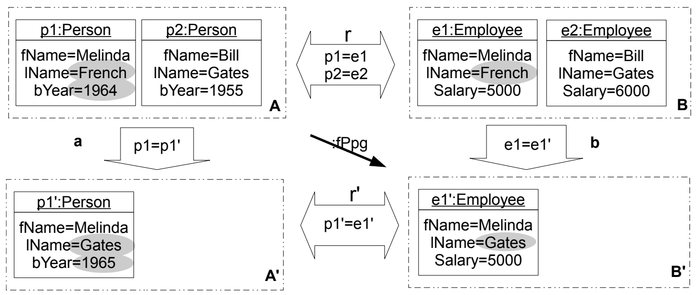


Fig. 4: Example of update propagation

A natural requirement for sd-lenses is that if the input delta changes nothing, the output delta should also change nothing. Formally, we call an sd-lens *stable* if the following law holds for any corr $r\colon A \to B$ (see Fig. 3c):

(IdPpg) $\quad$ fPpg(id$A$,r)=(id$B$,r) and bPpg(id$B$,r)=(id$A$,r).

The rest of the paper assumes this law holds by default unless the otherwise is explicitly specified.

We write an sd-lens over a triple space $\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$ as a double bidirectional arrow $\boldsymbol{\lambda}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ meaning that the second arrow refers to a pair of operations (fPpg, bPpg) constituting the lens.

3.2 Invertibility and undoability

A basic requirement for *bidirectional* model synchronization is compatibility of propagation operations between themselves. Given a corr $r\colon A \leftrightarrow B$, an update $a\colon A \to A'$ is propagated into update $b = a.\mathsf{fPpg}(r)$, which can be propagated back to update $a' = b.\mathsf{bPpg}(r)$. For an ideal situation of *strong invertibility*, we should require $a' = a$. Unfortunately, it does not hold in general because **A**-specific part of the information is lost in passing from a to b, and cannot be restored. For example, in Fig. 4 **A**-objects have birth years, which are absent on the **B**-side and hence are lost in a'. However, we could still require invertibility for data shared between A and B. In our example, name changes are shared and will be restored in a'; hence, $a \neq a'$ but $a'.\mathsf{fPpg} = a.\mathsf{fPpg}$. We thus come to the notion of *weak invertibility* of update propagation; it is formalized as follows.

Definition 4 (update equivalence) Given an sd-lens $\boldsymbol{\lambda}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ and a corr $r\colon A \leftrightarrow B$, two updates of model A, $a_1\colon A \to A'_1$ and $a_2\colon A \to A'_2$, are called *r-equivalent* if $a_1.\mathsf{fPpg}(r) = a_2.\mathsf{fPpg}(r)$; we then write $a_1 \sim_r a_2$. Similarly, we introduce *r-equivalence* $b_1 \sim_r b_2$ on **B**-side. (It is easy to see that both relations are indeed equivalence relations.)

Definition 5 (invertible lenses) Operations fPpg and bPpg are *(weakly) invertible* if equations below hold for any $r\colon A \leftrightarrow B$ and all $a\colon A \to A'$, $b\colon B \to B'$:

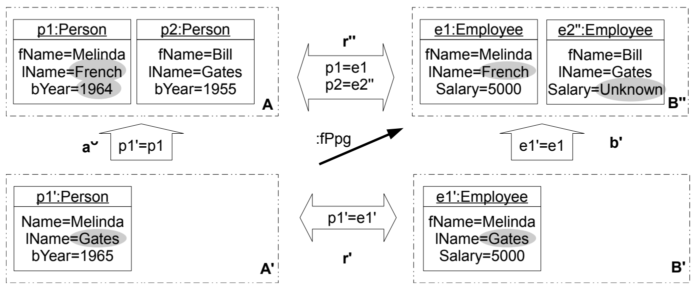


Fig. 5: Undoing update a from Fig. 4

(fbInv) $a.\mathsf{fPpg}(r).\mathsf{bPpg}(r) \sim_r a.$
(bfInv) $b.\mathsf{bPpg}(r).\mathsf{fPpg}(r) \sim_r b.$

We will call an sd-lens satisfying the laws *invertible*. We show in [10] that invertible sd-lenses can be implemented with triple-graph grammars.

Another important requirement for a reasonable BX is *undoability* discussed by Stevens [15] in the state-based setting. In an ideal situation of *strong undoability*, if update a is first propagated as b and then is cancelled by delta $a^{\smile}\colon A' \to A$, we require a reasonable BX to produce delta $b^{\smile}\colon B' \to B$ to cancel the change on the other side. Unfortunately, it does not hold in general because some information about B may be lost in B' and cannot be restored. For example, Fig. 5 continues the story of Fig. 4 and shows an update $a^{\smile}$ canceling a. According to corr r', a corresponding new object $e2$ (Bill Gates in $\mathbf{B}$) should be inserted into model B' and return it back to B. However, since Bill's Salary was lost in B', the propagation of $a^{\smile}$ along r' can only set his Salary to Unknown thus resulting in a new object $e2''$ and a new model B''. It is a vertical-delta analog of the phenomenon we have just discussed for horizontal deltas, and the strong condition should be again relaxed by considering updates up to their equivalence.

Definition 6 (undoable lenses) An sd-lens is called *(weakly) undoable* if the following *forward-undo* and *backward-undo* laws hold:
(fUndo) Let $(b, r') = \mathsf{fPpg}(a, r)$. Then $a^{\smile}.\mathsf{fPpg}(r') \sim_{r'} b^{\smile}$.
(bUndo) Let $(a, r') = \mathsf{bPpg}(b, r)$. Then $b^{\smile}.\mathsf{bPpg}(r') \sim_{r'} a^{\smile}$.

In the long version [7], we show that an sd-lens may be (i) invertible but not undoable, (ii) undoable but not invertible, or (iii) invertible and undoable. It means that the two notions are independent and consistent.

To unify terminology, we will call an invertible/undoable lens *horizontally/*resp. *vertically well-behaved (Wb).* A lens is *well-behaved* if it is both horizontally and vertically Wb. We will also refer to the laws as *horizontal/vertical round-tripping.*

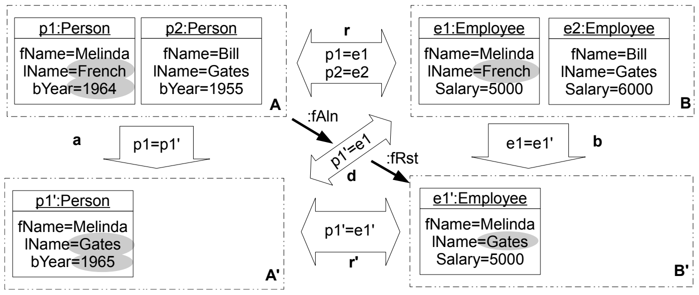


Fig. 6: Two steps in update propagation

4 Consistency maintenance and alignment

We have seen that a well-behaved sd-lens exhibits a truly BX-behavior. An advantage of the framework is its simplicity yet applicability to practical scenarios. However, simplicity of the sd-lens framework comes for a price.

First, an update propagation in sd-lenses actually consists of two steps, and their coupling prevents the reuse of operations in the implementation. Consider Fig. 6 that shows the case of propagation in Fig. 4 in more details. The first step is to align models A' and B and compute a new (diagonal) correspondence delta $d\colon A' \leftrightarrow B$ based on the original delta r and update $a\colon A \to A'$. We call this operation *forward (re-)alignment* and denote it as fAln. Note that re-alignment is nothing but composition of two deltas (a simple computation), and should not be confused with delta discovery (requiring heuristics). With this reservation, we will call re-alignment just alignment.

The new correspondence d reveals an inconsistency: objects $p1'$ and $e1$ are declared to be the same yet their lName attributes are different. Hence, in the second step consistency must be restored by updating object $e1$ to $e1'$, and thus we produce an update delta $b\colon B \to B'$ and consistent correspondence delta $r'\colon A' \leftrightarrow B'$ from delta d. We call this operation *forward (consistency) restoration*, fRst. Since different restoration operations can be built on top of the same alignment framework, we could reuse alignment operations. However, their reuse cannot be realized within the sd-lens interface, since (re-)alignment operations are woven into update propagation in sd-lenses.

The second problem of the sd-lens interface is related to an important BX requirement — Hippocraticness law of Meertens/Stevens [13, 15]. When model A is updated to A', it may happen that the new diagonal delta d is still consistent and then nothing should be done on the **B**-side. However, since in sd-lenses we have no access to diagonal deltas, we cannot formulate the requirement above.

We call a pair of forward and backward alignment operations an *alignment framework* to stress its basic supporting role for restoration operations built on top of it. We call a pair of forward and backward restoration operations a *maintainer*. Below in this section we formalize the two notions and show that well-behaved maintainers correctly implement well-behaved sd-lenses.

$$\begin{tikzcd}
\bullet \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, phantom, "\Searrow{:}\mathsf{fAln}"] & \bullet \\
\bullet \arrow[ur, dashed, leftrightarrow, "a*r"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\bullet \arrow[r, leftrightarrow, "r"] \arrow[dr, dashed, leftrightarrow, "r*b"'] \arrow[dr, phantom, "\Swarrow{:}\mathsf{bAln}"] & \bullet \arrow[d, "b"] \\
& \bullet
\end{tikzcd}
\qquad
\begin{tikzcd}
\bullet \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, dashed, leftrightarrow, "r*b"] & \bullet \arrow[d, "b"] \\
\bullet \arrow[r, dashed, leftrightarrow, "r'"'] \arrow[ur, dashed, leftrightarrow, "a*r"] & \bullet
\end{tikzcd}$$

(a) fAln (b) bAln (c) fAln;bAln = bAln;fAln

Fig. 7: Alignment operations and their laws

4.1 Alignment taken seriously

We define the notion of alignment framework as a triple space enriched with re-alignment operations.

Definition 7 (Alignment framework) An *alignment framework* over a triple space $\mathbf{R}\colon \mathbf{A} \leftrightarrow \mathbf{B}$ is a couple of operations

$$\mathsf{fAln}\colon \mathbf{\Delta_A} \boxtimes \mathbf{\Delta_{AB}} \to \mathbf{\Delta_{AB}} \qquad \text{and} \qquad \mathsf{bAln}\colon \mathbf{\Delta_{AB}} \leftarrow \mathbf{\Delta_B} \boxtimes \mathbf{\Delta_{AB}}$$

called *forward* and *backward alignment* resp., where symbols $\boxtimes$ denote subsets of the respective Cartesian products consisting of all incident arrows as specified by Fig. 7(a,b) (see p.7). We will also write $a*r$ for $\mathsf{fAln}(a,r)$ and $r*b$ for $\mathsf{bAln}(b,r)$.

There are two laws. Identity updates do not actually need re-alignment:
(IdAln) $\quad \mathsf{id}A * r = r = r * \mathsf{id}B$
for any corr $r\colon A \to B$.

The result of applying a sequence of interleaving forward and backward alignments does not depend on the order of application as shown in Fig. 7(c):
(AlnAln) $\quad (a*r)*b = a*(r*b)$
for any $a \in \mathbf{\Delta_A}, r \in \mathbf{\Delta_{AB}}, b \in \mathbf{\Delta_B}$.

We will write an alignment framework as an arrow $\boldsymbol{\alpha}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$.

4.2 Consistency maintainers: Hippocratic update propagation

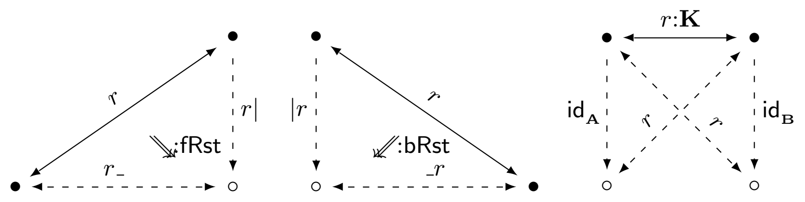
Definition 8 (maintainers) A *(consistency) maintainer* over an alignment framework $\boldsymbol{\alpha}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ comprises (i) a subclass $\mathbf{K} \subset \mathbf{\Delta_{AB}}$ of *consistent* corrs and (ii) a couple of *consistency restoration* operations

$$\mathsf{fRst}\colon \mathbf{\Delta_{AB}} \to \mathbf{\Delta_B} \boxtimes \mathbf{\Delta_{AB}} \qquad \text{and} \qquad \mathsf{bRst}\colon \mathbf{\Delta_A} \boxtimes \mathbf{\Delta_{AB}} \leftarrow \mathbf{\Delta_{AB}}$$

of arities shown in Fig. 8 (a,b): output nodes and arrows are shown blank and dashed resp.

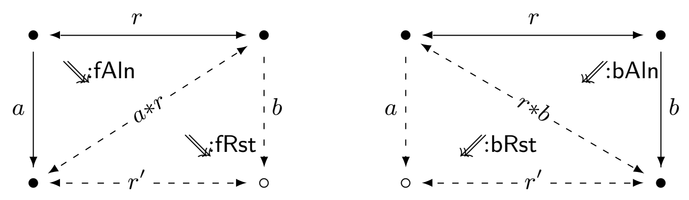
If $(b, r') = \mathsf{fRst}(r)$, we will also write $r|$ for b and $r_{\_}$ for r'; similarly, if $(a, r') = \mathsf{bRst}(b)$, we write $|r$ and ${}_{\_}r$ for a and r'. In composed formulas, bars and underscores always have the highest priority.

A maintainer is called *correct* if its output corrs are always consistent, and are compositions of the original corr with output updates:
(Corr) $\quad r * r| = r_{\_} \in \mathbf{K} \quad \text{and} \quad |r * r = {}_{\_}r \in \mathbf{K}$



(a) fRst (b) bRst (c) Hippocraticness

Fig. 8: Consistency restoration operations (a,b) and their laws



(a) Definition of fPpg (b) Definition of bPpg

Fig. 9: From maintainers to lenses

A maintainer is called *Hippocratic* (we borrow Stevens' term [15]) if it does nothing for an originally consistent corr as shown in Fig. 8(c):
(Hipp) If $r\colon A \to B \in \mathbf{K}$, then $|r = \mathsf{id}A$, $r| = \mathsf{id}B$ and $_-r = r = r_-$.

We write a maintainer as an arrow $\boldsymbol{\mu}\colon \mathbf{A} \overset{\mathbf{K}\subset\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ comprising pairs of operations (fAln,bAln) and (fRst,bRst) over the triple space $\mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$.

4.3 From maintainers to lenses: invertibility and undoability

Maintainers are designed to implement lenses: update propagation operations can be defined via alignment and restoration operations as shown in Fig. 9(a,b).

Definition 9 (from maintainers to lenses) Given a correct maintainer $\boldsymbol{\mu}\colon \mathbf{A} \overset{\mathbf{K}\subset\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$, we define a lens $\ulcorner\boldsymbol{\mu}\urcorner\colon \mathbf{A} \overset{\mathbf{K}}{\Longleftrightarrow} \mathbf{B}$ by setting
$\mathsf{fPpg}(a,r) \overset{\text{def}}{=} (d|, d_-)$ with $d = a * r$, and $\mathsf{bPpg}(b,r) \overset{\text{def}}{=} (|e, {_-e})$ with $e = r * b$.

It is easy to see that lens $\ulcorner\boldsymbol{\mu}\urcorner$ is stable as soon as $\boldsymbol{\mu}$ is Hippocratic. That is, a correct and Hippocratic maintainer implements a stable lens.

Now we want to state conditions for $\boldsymbol{\mu}$ ensuring that the lens $\ulcorner\boldsymbol{\mu}\urcorner$ is well-behaved. Since the notion of update equivalence is crucial here, we first reformulate it as *corr equivalence* in terms of restoration operations.

Definition 10 (corr equivalence) Two corrs with the same target, $r_i\colon A_i \leftrightarrow B$, $i = 1,2$ are called *forward equivalent* if $r_1| = r_2|$; we write $r_1 \sim_\bullet r_2$. Dually, two corrs with the same source $r_i\colon A \leftrightarrow B_i$ are *backward equivalent*, $r_1 \,{}_\bullet\!\!\sim r_2$, if $|r_1 = |r_2$.

The next step is to substitute operations defined in Definition 9 into Definitions 5 and 6 of invertibility and undoability.

Definition 11 (well-behaved maintainer) (a) A correct maintainer is called *invertible* or *horizontally well-behaved (hWb)* if the following two dual conditions hold for any $r\colon A \leftrightarrow B \in \mathbf{K}$:

(fbInv_m) For any $a\colon A \to A'$, let $d1 = a{*}r$, $e1 = r * d1|$. Then $|e1 * r \sim_\bullet d1$

(bfInv_m) For any $b\colon B \to B'$, let $d1 = r{*}b$, $e1 = |d1 * r$. Then $r * e1| \,{}_\bullet\!\!\sim d1$

(b) A correct maintainer is called *undoable* or *vertically well-behaved (vWb)* if the following two dual conditions hold for any $r\colon A \leftrightarrow B \in \mathbf{K}$:

(fUndo_m) For any $a\colon A \to A'$, let $d1 = a{*}r$, $b = d1|$, $r' = d1_{\_}$, $d2 = r' * b^{\smile}$, and $e2 = a^{\smile} * r'$. Then $d2 \,{}_\bullet\!\!\sim r' * e2|$

(bUndo_m) For any $b\colon B \to B'$, let $d1 = r{*}b$, $a = |d1$, $r' = {}_{\_}d1$, $d2 = a^{\smile} * r'$, and $e2 = r' * b^{\smile}$. Then $d2 \sim_\bullet |e2 * r'$

Details clarifying the meaning of formulas can be found in the long version. The notion of invertible maintainer is implicit in [10], where alignment and restoration operations are realized by TGG-means.

(c) A correct maintainer is called *well-behaved (Wb)* if it is well-behaved both horizontally and vertically.

Theorem 1. *Let* $\boldsymbol{\mu}\colon \mathbf{A} \overset{\mathbf{K}\subset\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ *be a correct maintainer and* $\ulcorner\boldsymbol{\mu}\urcorner\colon \mathbf{A} \overset{\mathbf{K}}{\Longleftrightarrow} \mathbf{B}$ *is the sd-lens derived from it. Then the following holds*

(i) $\ulcorner\boldsymbol{\mu}\urcorner$ *is stable iff* $\boldsymbol{\mu}$ *is Hippocratic.*

(ii) $\ulcorner\boldsymbol{\mu}\urcorner$ *is invertible iff* $\boldsymbol{\mu}$ *is invertible.*

(iii) $\ulcorner\boldsymbol{\mu}\urcorner$ *is undoable iff* $\boldsymbol{\mu}$ *is undoable.*

Hence, a correct maintainer $\boldsymbol{\mu}$ *implements a Wb sd-lens* $\ulcorner\boldsymbol{\mu}\urcorner$ *iff* $\boldsymbol{\mu}$ *is itself Wb.*

The proof of the theorem can be found in the long version. The theorem shows that heavy definitions of maintainers' laws can be hidden under the hood of the sd-lens framework. The latter thus demonstrates a reasonable trade-off between concreteness and abstraction: it is abstract enough to free the user from the (re-)alignment concerns, yet provides enough flexibility by explicitly including deltas.

5 Related Work

Algebraic frameworks for symmetric BX did not get as much attention as asymmetric ones, perhaps, because of technical difficulties of working in the symmetric situation. Several closely related state-based frameworks were built by Meertens [13], Stevens [15], and Diskin [6]. In these frameworks, model consistency is a binary relation on model spaces. For us, consistency is a property of the correspondence between models (the idea first proposed in [4]). State-based frameworks mentioned above appear as special cases of our delta-base maintainers, if deltas are merely pairs of models (we call such triple spaces *simple*).

Then identity, update inversion and alignment operations are trivial: $\mathsf{id}A = AA$, $(AA')^{\smile} = A'A$, $AA' * AB = A'B$, $AB * BB' = AB'$, and are uniquely determined by model spaces. Hence, these operations can be removed from the signature and we come to the state-based setting. If undoability of [15] is reshaped to its weak form, then Stevens' coherent transformations are exactly our vertically Wb maintainers over simple triple spaces. If invertibility is also reshaped to its weak form, then undoable and invertible trigonal systems of [6] are exactly our Wb maintainers over simple triple spaces. Precise results can be found in [7].

A different state-based algebraic model of symmetric BX is symmetric lenses with complement by Hofmann *et al* [11] (*ssc-lenses*). They can be seen as our sd-lenses over simple model spaces (update deltas are pairs) but non-simple correspondences, that is, we still consider a *set* $\mathbf{R}(A, B)$ of corrs for a given pair of models (A, B). Given a model A' and a corr $r\colon A \leftrightarrow B$, we can simulate ssc-lens operation $\mathsf{putr}(A, r)$ by computing $\mathsf{fPpg}(AA', r)$; symmetrically for B' and r. Then laws called round-tripping in [11] and our IdPpg laws coincide; however, our invertibility (which we believe is truly about round-tripping) and undoability laws are not considered in [11]. On the other hand, symmetric lenses by Hofmann *et al* have an element *missing* referring to minimal models (empty ones, if permitted by the metamodels), which is omitted in sd-lenses. To fill-in the gap, we need to enrich our model spaces with *initial objects* (a construct well-known in category theory); we leave it for future work.

Mathematical foundations for building delta-based frameworks (called *tile algebra*) are described in [4]. Diagonal synchronizers specified there are basically sd-lenses that distinguish between consistent and inconsistent corrs at the input of propagation operations; in addition, they are equipped with alignment operations called rematching. However, neither update inversion, nor the round-tripping laws are considered in [4].

6 Conclusion

A delta-based symmetric BX is a synchronization module that does nothing but propagating vertical deltas over horizontal ones; how these deltas are computed and passed to the module is a separate concern. This design provides a flexible architecture and fixes compositional problems of the state-based frameworks. In the paper we built two algebraic frameworks for symmetric delta-based BXs: more abstract sd-lenses that screen simple but tedious re-alignment computations from the user, and closer to implementation maintainers. We found new— weaker—versions of important invertibility and undoability laws, which do constrain synchronization behavior, and yet do not exclude many practically interesting BXs incompatible with the strong laws considered previously. Our main result shows that an sd-lens can be implemented by a suitable maintainer, and the former is weakly invertible and undoable iff the latter is such.

The framework still lacks lens and maintainer combinators for specifying complex BX in a compositional way. A well-designed set of combinators would make our frameworks practically applicable to the design of BX languages. We leave it for future work.

Acknowledgment. We are grateful to Michał Antkiewicz, Leo Passos and Arif Wider for discussion and fruitful comments. Financial support was provided by the Ontario Research Fund and NSERC.

References

1. Alanen, M., Porres, I.: Difference and union of models. In: UML. pp. 2–17 (2003)
2. Barbosa, D.M.J., Cretin, J., Foster, N., Greenberg, M., Pierce, B.C.: Matching lenses: alignment and view update. In: ICFP. pp. 193–204 (2010)
3. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.F.: Bidirectional transformations: A cross-discipline perspective. In: ICMT (2009)
4. Diskin, Z.: Model synchronization: mappings, tile algebra, and categories. In: Postproc. GTTSE 2009. pp. 92–165 (2011)
5. Diskin, Z., Xiong, Y., Czarnecki, K.: From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. Journal of Object technology 10, 6:1–25 (2011)
6. Diskin, Z.: Algebraic models for bidirectional model synchronization. In: MoDELS. pp. 21–36 (2008)
7. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From state- to delta-based bidirectional model transformations: the symmetric case. Tech. Rep. GSDLAB-TR 2011-05-03, GSD Lab, University of Waterloo (2011), http://gsd.uwaterloo.ca/node/338
8. Foster, J.N., Greenwald, M., Moore, J., Pierce, B., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. 29(3) (2007)
9. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and System Modeling 8(1), 21–43 (2009)
10. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on TGG. In: MODELS (2011)
11. Hofmann, M., Pierce, B.C., Wagner, D.: Symmetric lenses. In: POPL (2011)
12. Hu, Z., Mu, S.C., Takeichi, M.: A programmable editor for developing structured documents based on bidirectional transformations. Higher-Order and Symbolic Computation 21(1-2), 89–118 (2008)
13. Meertens, L.: Designing constraint maintainers for user interaction (1998), Available from http://www.kestrel.edu/home/people/meertens/
14. Song, H., Huang, G., Chauvel, F., Zhang, W., Sun, Y., Mei, H.: Instant and incremental QVT transformation for runtime models. In: MODELS (2011)
15. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Software and System Modeling 9(1), 7–20 (2010)
16. Xing, Z., Stroulia, E.: UMLDiff: an algorithm for object-oriented design differencing. In: ASE. pp. 54–65 (2005)
17. Xiong, Y., Liu, D., Hu, Z., Zhao, H., Takeichi, M., Mei, H.: Towards automatic model synchronization from model transformations. In: ASE. pp. 164–173 (2007)
18. Xiong, Y., Hu, Z., Zhao, H., Song, H., Takeichi, M., Mei, H.: Supporting automatic model inconsistency fixing. In: ESEC/SIGSOFT FSE. pp. 315–324 (2009)
19. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Software and Systems Modeling. To appear