Writing Bidirectional Model Transformations as Intentional Updates

Tao Zan
The Graduate University for Advanced Studies
Tokyo, Japan
zantao@nii.ac.jp

Hugo Pacheco
National Institute of Informatics
Tokyo, Japan
hpacheco@nii.ac.jp

Zhenjiang Hu
National Institute of Informatics
Tokyo, Japan
hu@nii.ac.jp

ABSTRACT

Model synchronization plays an important role in model-driven software development. *Bidirectional model transformation* approaches provide techniques for developers to specify the bidirectional relationship between source and target models, while keeping related models synchronized for free. Since models of interest are usually not in a one-to-one correspondence, this synchronization process is inherently ambiguous. Nevertheless, existing bidirectional model transformation tools focus mainly on enforcing consistency and provide developers only limited control over how models are synchronized, solving the latent ambiguity via default strategies whose behavior is unclear to developers. In this paper, we propose a novel approach in which developers write *update programs* that succinctly describe how a target model can be used to update a source model, such that the bidirectional behavior is fully determined. The new approach mitigates the unpredictability of existing solutions, by enabling a finer and more transparent control of what a bidirectional transformation does, and suggests a research direction for building more robust bidirectional model transformation tools.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classifications—*Specialized application languages*

General Terms

Languages, Design

Keywords

Bidirectional transformation, model-driven development, update language, model sychronization

1. INTRODUCTION

The Model-Driven Engineering (MDE) perspective to software development focuses on models as the primary development artifact. In the same model-driven ecosystem, various models may naturally coexist to express different but interrelated concepts (like different views of the same entity or entities at different levels of abstraction), and transforming between these loosely similar models is a typical problem in MDE. Ideally, such transformations should be bidirectional (or multi-directional), in the sense that a single artifact specifies how we can mutually transform between pairs of models in order to keep them consistent.

Over the last 10 years, a few bidirectional model transformation [5, 2] approaches have been proposed to facilitate the consistent development of models. The most popular approaches in MDE are the OMG's Query/View/Transformation (QVT) standard [11] —in particular the QVT Relations (QVT-R) language— and Triple Graph Grammars (TGGs) [13], that prescribe writing a bidirectional transformation as a *declarative consistency relation* between two meta-models. Other *transformational* approaches consider writing a standard unidirectional transformation as the consistency relation, and deriving a suitable backward transformation, as followed by SyncATL [15] or GRoundTram [6].

As most interesting examples of bidirectional transformations are not bijective [14], there may be multiple ways to transform two models into a consistent state, introducing ambiguity. Despite of this fact, existing bidirectional model transformation tools focus mainly on enforcing consistency, and each considers only one particular strategy (out of a myriad possible) to translate a non-deterministic specification into an actual bidirectional transformation procedure, of which developers have little or no control.

The lack of a clear (deterministic) semantics has been one of the major issues hindering the serious practical adoption of bidirectional transformation tools. Quoting Stevens [14]:

> The developer needs full control of what the transformation does. [...] We claim that determinism is necessary in order to ensure, first, that developers will find tool behaviour predictable, and second, that organisations will not be unacceptably "locked in" to the tool they first use.

In this paper, we propose a novel *programming by update* approach to define a wide class of bidirectional model transformations that have a flow of information in a particular direction. In our approach, developers write update programs that describe how to embed a target model as an update on a source model. An update in our sense allows to express the relationship between source and target models in a simple way, as in the relational approach, combined with additional actions that supply the missing pieces to tame the ambiguity in bidirectional behavior. In sharp con-

ICSE '14, May 31 - June 7, 2014, Hyderabad, India

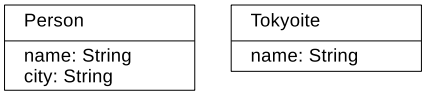

Figure 1: Class diagrams of the `Person` and `Tokyoite`.

trast to existing approaches, our approach supports a more predictable bidirectional programming style that features a finer and more transparent control of what a bidirectional transformation does.

2. A MOTIVATING EXAMPLE

As a very simple example of a typical bidirectional model transformation, imagine that we have a source set of people, each person having a name and a city where he/she lives, and a target set of tokyoites, each consisting of the name of a person living in Tokyo. The corresponding source and target meta-models are represented in Figure 1. We can easily write a QVT-R transformation (though it would be better called a consistency relation) that specifies the relationship between people and tokyoites:

```
transformation p2t (people:People, tokyoites:Tokyoites){
 top relation person2tokyoite {
   pn : String;
   domain people person:Person
     { name = pn, city = 'Tokyo'};
   domain tokyoites tokyoite:Tokyoite { name = pn};
}}
```

This QVT-R bidirectional transformation reads that for each person living in Tokyo in the source domain there exists a tokyoite with the same name, and vice-versa.

Even for this simplistic example, the consistency relation is deterministic in one of the directions (for each set of people there is exactly one set of tokyoites) but it is not bijective, since there may be arbitrarily many sets of people related with the same set of tokyoites, namely all those with the same people living in Tokyo.

Therefore, the behavior of the concrete bidirectional transformation that is derived from this specification will depend on the underlying bidirectionalization strategy used by each particular tool. For instance, the backward transformation inferred by EMoflon [1] (TGGs) would ignore current source people and generate a new set of people afresh from a new set of tokyoites; Echo [10] (QVT-R) would return a non-deterministic choice between all consistent source sets at a minimal distance to the current source set under some distance measure; while SyncATL or GRoundTram would try to propagate the modifications made to the view set of tokyoites as inferred replace/insert/delete operations on the current source set of people[1].

Unfortunately, the results provided by existing bidirectional model transformation tools often come as a (unsatisfactory) surprise to the developer that specifies a bidirectional transformation. This is because, independently of the tool that is used, he/she can not clearly understand nor completely control what the bidirectional transformation does.

[1]There exist many other bidirectional model transformation tools, including other implementations of TGGs and QVT-R and miscellaneous bidirectional programming languages.

3. WRITING INTENTIONAL UPDATES

In this section, we illustrate our novel bidirectional model transformation approach of writing bidirectional updates on a source meta-model to reflect the content of a target meta-model, and exemplify how our language can naturally express different intentional bidirectional synchronization strategies.

We will assume that models are concretely encoded in XMI, the standard XML interchange convention for representing models prescribed by the OMG.

3.1 Multiple update strategies

In our proposed language, we can write the previous QVT-R bidirectional transformation as the following update. (A more detailed description of the syntax and semantics of our language will be given in the next section.)

```
PROCEDURE p2t($people:People, $tokyoites:Tokyoites) =
UPDATE $person IN $people/Person BY
{
  UNMATCHV -> CREATE VALUE
          <Person name='' city='Tokyo' </Person>
}
FOR VIEW $tokyoite IN $tokyoites/Tokyoite
MATCHING BY @name
WHERE @city = 'Tokyo'
```

The intuition of this update is similar to the QVT-R relation — for each person in a source domain, match it with a view tokyoite if the person lives in Tokyo— but there are a few diverging nuances. The first is that it has particular direction —update a source using a view— what motivates the wording "view" instead of "target" and plays a crucial role in the design of our language. Second, it also reads to create new persons living in Tokyo for tokyolites that do not have matching persons. Third, instead of the relational "for all there exists" semantics, matching conditions have a more procedural meaning and can be understood as keys that guide the synchronization of source and view elements.

These three details support the claim that our update language allows users to intuitively control the behavior of a bidirectional transformation; the design of our language guarantees that the forward transformation is deterministic, and "matching"/"unmatching" conditions solve the ambiguity for the backward transformation.

To understand more precisely the meaning of our update, consider two (inconsistent) source and view models:

```
<xmi:XMI xmlns="People">
       <Person name="Hugo" city="Tokyo"/>
       <Person name="Sebastian" city="Kiel"/>
       <Person name="Zhenjiang" city="Tokyo"/>
</xmi:XMI>


<xmi:XMI xmlns="Tokyoites">
       <Tokyoite name="Zhenjiang" />
       <Tokyoite name="Tao" />
</xmi:XMI>
```

that we want to synchronize by reflecting the view information as a source update. Applying our update, we get the following updated source that is consistent with the view.

```
<xmi:XMI xmlns="People">
       <Person name="Sebastian" city="Kiel"/>
       <Person name="Zhenjiang" city="Tokyo"/>
       <Person name="Tao" city="Tokyo"/>
</xmi:XMI>
```

This update keeps Sebastian that didn't live in Tokyo in the original source, but deletes Hugo that lived in Tokyo in order to ensure that the updated source is consistent with the view tokyoites.

Even being reasonable for many situations, this strategy is not the only possible one; for example, if the user of the view works for the Tokyo municipal council, deleting people from the view may actually mean that such person just moved to another city instead of disappearing from the source database. We can easily describe this alternative update strategy by adding an additional unmatching condition to our update:

```
PROCEDURE p2t'($people:People, $tokyoites:Tokyoites) =
UPDATE $person IN $people/Person BY {
  UNMATCHV -> CREATE VALUE
          <Person name='' city='Tokyo' </Person>
| UNMATCHS -> KEEP $person;
            REPLACE IN $person/city WITH 'Ithaca'
}
FOR VIEW $tokyoite IN $tokyoites/Tokyoite
MATCHING BY @name
WHERE @city = 'Tokyo'
```

Running this second update moves people like Hugo to a new city, in this case Ithaca, producing the updated source:

```
<xmi:XMI xmlns="People">
       <Person name="Hugo" city="Ithaca"/>
       <Person name="Sebastian" city="Kiel"/>
       <Person name="Zhenjiang" city="Tokyo"/>
       <Person name="Tao" city="Tokyo"/>
</xmi:XMI>
```

3.2 Bidirectional semantics of updates

Despite the emphasis of our approach is on writing updates, update programs in our proposed language have a bidirectional interpretation. For an update program, we derive an *update function* $U(s, v') = s'$, that applies an update to a source s so that the updated source s' is consistent with a given view v', and a *query function* $Q(s) = v$, that computes a consistent view v for a given source s.

For our two sample update programs, the derived query function will be the same —filter only the people living in Tokyo as tokyoites— because they only differ for unmatched source people that lived in Tokyo. Such query is semantically equivalent to the following XQuery code.

```
<xmi:XMI xmlns="Tokyoites">
  { for $person in doc("people.xml")/xmi:XMI/Person
      where $person/@city = 'Tokyo'
      return <Tokyoite name="{$person/@name}"/> }
</xmi:XMI>
```

To guarantee that a query Q conveys the notion of consistency between sources and views, our language is carefully designed to ensure that the consistency relation is deterministic, so that modelling it by a function is appropriate. Moreover, the bidirectional semantics of our language satisfies two basilar synchronization properties; that an update U consistently embeds view information to the source:

$$U(s, v') = s' \;\Rightarrow\; Q(s') = v'$$

and that it does not update already consistent sources:

$$Q(s) = v \;\Rightarrow\; U(s, v) = s$$

These two properties are commonly found for view updating in the database community and for bidirectional programming in the programming languages community [5].

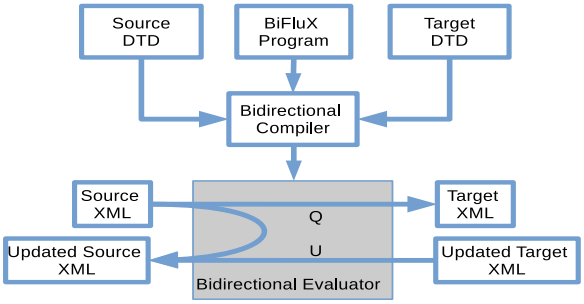

Figure 2: Our update-based bidirectional model transformation framework.

4. LANGUAGE AND FRAMEWORK

The language that we have presented so far implicitly assumes a concrete XML-centric representation of models and meta-models. This is because our design is not particularly restricted to model-driven bidirectional transformations, but can be widely used for a variety of data that is nowadays stored in XML formats. Models can be naturally represented as XML, especially using XMI, as prescribed for Ecore models in the Eclipse Modeling Framework (EMF).

Instead of writing a new language from scratch, we attempt to integrate our approach with existing work on XML database updates. As a good candidate (but not essential to our approach), we choose to base our design on FLUX [4], a simple but well-structured XML update language. More precisely, we extend the syntax of FLUX, suitable for defining in-place updates on sources, to accommodate an additional notion of view. This is done with a new kind of non-in-place "update-for-view" update that synchronizes two source-view sequences; such synchronization can be configured by the developer via a set of matching/unmatching clauses that describe actions for individual source-view elements. The syntax of our bidirectional update language, named BIFLUX, is shown in Figure 3: *Stmt* denotes general statements and *Upd* individual updates; *Path* and *Expr* are arbitrary XPath paths and XQuery expressions; and *Pat* stands for XDuce-style patterns with variables bounded by regular expression types [7].

Figure 2 depicts the architecture of our update-based bidirectional model transformation framework. A BIFLUX update program is evaluated in two stages. On a first stage, it is statically compiled against a source and a view meta-model (represented as DTDs), to produce a bidirectional executable. The generated executable can then be evaluated for particular models conforming to the DTDs: either as a query Q, that computes a consistent view model from a source model, or as an update U that updates a source model to be consistent with given a view model.

5. CURRENT STATUS AND IMPACT

In this paper, we have proposed a novel update-based approach for bidirectional model transformations. Our framework has been fully implemented, and the current prototype

```
Stmt    ::=  Upd [WHERE Expr]
         |   IF Expr THEN Stmt ELSE Stmt
         |   Stmt ; Stmt
         |   LET Pat := Expr IN Stmt
         |   CASE Expr OF { Cases }
         |   { Stmt }
         |   PROCEDURE Name '(' Path, ..., Path ')'
Upd     ::=  INSERT (BEFORE | AFTER) PathPat VALUE Expr
         |   INSERT AS (FIRST | LAST) INTO PathPat VALUE Expr
         |   DELETE [FROM] PathPat
         |   KEEP Path
         |   CREATE VALUE Expr
         |   REPLACE [IN] PathPat WITH Expr
         |   UPDATE PathPat BY Stmt
         |   UPDATE PathPat BY VStmt FOR VIEW PathPat [Match]

Cases   ::=  Pat → Stmt
         |   Cases '|' Cases
VStmt   ::=  VUpd '|' VUpd
         |   { VUpd }
VUpd    ::=  MATCH → Stmt
         |   UNMATCHS → Stmt
         |   UNMATCHV → Stmt
Match   ::=  MATCHING BY Path
         |   MATCHING SOURCE BY Path
         |                VIEW BY Path
PathPat ::=  [Pat IN] Path
```

Figure 3: Concrete syntax of BiFluX.

is available in our website[2]. The implementation ensures that BiFluX programs and underlying XML models are correct against the DTD meta-model specifications, and defines the bidirectional semantics for compiled programs by translating them into *putlenses* [12], a powerful combinator library for bidirectional programming over general tree data structures.

To assess the expressiveness of our language, we have successfully tested it with some trademark bidirectional model transformation use cases (also available online):

Bookmarks A bidirectional conversion between the popular Netscape and XBEL bookmark formats [9].

Nested Sections A bidirectional mapping between Wiki formats with multiple levels of nested headings [3].

Address Books A bidirectional transformation between address book formats that handles inter-model dependencies and duplicated structure [8].

To the best of our knowledge, we have made the first proposal for an update-based bidirectional model transformation framework. The new approach promises practical impact, in that describing bidirectional transformations as updates forecasts a nice tradeoff between the elegant declarative style of relational approaches and the urging need for fine and transparent specification mechanisms that prevent unpredictable bidirectional behavior. Our success in unifying diverse bidirectional model transformation use cases in a single update language is a promising result, that we plan to explore further in the near future with a more realistic model-driven development scenario. A deeper integration with existing model-driven development processes and added support for graph-like structures are also vital for more serious applications. We believe that this research direction can help fostering the acceptance and development of more robust bidirectional model transformation tools.

6. REFERENCES

[1] A. Anjorin, M. Lauder, S. Patzina, and A. Schürr. eMoflon: Leveraging EMF and Professional CASE Tools. *Informatik*, page 281, 2011.

[2] M. Antkiewicz and K. Czarnecki. Design space of heterogeneous synchronization. In *GTTSE 2007*, volume 5235 of *LNCS*, pages 3–46. Springer, 2008.

[3] D. M. J. Barbosa, J. Cretin, J. N. Foster, M. Greenberg, and B. C. Pierce. Matching lenses: alignment and view update. In *ICFP 2010*, pages 193–204. ACM, 2010.

[4] J. Cheney. Flux: Functional Updates for XML. In *ICFP 2008*, pages 3–14. ACM, 2008.

[5] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. In *ICMT 2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.

[6] S. Hidaka, Z. Hu, H. Kato, and K. Nakano. A compositional approach to bidirectional model transformation. In *ICSE-Companion 2009*, pages 235–238. IEEE, 2009.

[7] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. In *ICFP 2000*, pages 11–22. ACM, 2000.

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

[9] S. Kawanaka and H. Hosoya. biXid: a bidirectional transformation language for XML. In *ICFP 2006*, pages 201–214. ACM, 2006.

[10] N. Macedo, T. Guimaraes, and A. Cunha. Model Repair and Transformation with Echo. In *ASE 2013*. To appear, 2013.

[11] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation, Version 1.1. http://www.omg.org/spec/QVT/1.1/PDF/, 2011.

[12] H. Pacheco, Z. Hu, and S. Fischer. Monadic combinators for "putback" style bidirectional programming. In *PEPM 2014*. To appear, 2014.

[13] A. Schürr. Specification of graph translators with triple graph grammars. In *WG 1995*, volume 903 of *LNCS*, pages 151–163. Springer, 1995.

[14] P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software & Systems Modeling*, 9(1):7–20, 2010.

[15] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei. Towards automatic model synchronization from model transformations. In *ASE 2007*, pages 164–173. ACM, 2007.

[2] `http://www.prg.nii.ac.jp/projects/BiFluX`