Bidirectional model transformations in QVT: semantic issues and open questions

Perdita Stevens*

School of Informatics, University of Edinburgh

Abstract. We consider the OMG's Queries, Views and Transformations (QVT) standard as applied to the specification of bidirectional transformations between models. We discuss what is meant by bidirectional transformations, and the model-driven development scenarios in which they are needed. We analyse the fundamental requirements on tools which support such transformations, and discuss some semantic issues which arise. We argue that a considerable amount of basic research is needed before suitable tools will be fully realisable, and suggest directions for this future research.
Keywords: bidirectional model transformation, QVT

1 Introduction

The central idea of the OMG's Model Driven Architecture is that human intelligence should be used to develop models, not programs. Routine work should be, as far as possible, delegated to tools: the human developer's intelligence should be used to do what tools cannot. To this end, it is envisaged that a single platform independent model (PIM) might be created and transformed, automatically, into various platform specific models (PSMs) by the systematic application of understanding concerning how applications are best implemented on each specific platform. The OMG's Queries, Views and Transformations (QVT) standard [12] defines languages in which such transformations can be written.

In this paper we will discuss *bidirectional* transformations, focusing on basic requirements which such transformations should satisfy.

The structure of the paper is as follows. In the remainder of this section, we motivate bidirectional transformation, and especially, the need for non-bijective bidirectional transformations; we then discuss related work. Section 2 briefly summarises the most relevant aspects of the QVT standard. Section 3 discusses key semantic issues that arise. Section 4 proposes and motivates a framework and a definition of "coherent transformation". Finally Section 5 concludes.

In order to justify the considerable cost of developing a model transformation, it should ideally be reused; perhaps a vendor might sell the same transformation to many customers. However, in practice a transformation will usually have to be adapted to the needs of a particular application. Similarly, whilst we might

* Email: perdita@inf.ed.ac.uk. Fax: +44 131 667 7209

like to argue that only the PIM would ever need to be modified during development, with model transformation being treated like compilation, the transformed model never being directly edited, nevertheless in practice it will be necessary for developers to modify both the source and the target models of a transformation and propagate changes in both directions. The interesting albeit unfinished document [14] makes these and other points, emphasising especially that bidirectional transformations are a key user requirement on QVT, and that ease of use of the transformation language is another key requirement.

Even in circumstances where it is in principle possible to make every change in a single source model, and roll changes down to target models by reapplying unidirectional transformations, in practice this is not desirable for a number of reasons. A human reason is that different developers are familiar with different models, and even different modelling languages. Developers are less likely to make mistakes if they change models they are comfortable with. A technical reason is that some changes are most simply expressed in the vocabulary, or with respect to the structure, of one model. For example, a single change to one model might correspond, semantically, to a family of related changes in the other.

Given the need for transformations to be applied in both directions, there are two possible approaches: write two separate transformations in any convenient language, one in each direction and ensure "by hand" that they are consistent, or use a language in which one expression can be read as a transformation in either direction. The second is very much preferable, because the checking required to ensure consistency of two separate transformations is hard, error-prone, and likely to cause a maintenance problem in which one direction is updated and the other not, leaving them inconsistent. QVT Relational takes this second approach: a transformation written in QVT Relational is intended to be able to be read as a specification of a relation which should hold between two models, or as a transformation function in either direction.

1.1 Bidirectional versus bijective transformations

A point which is vital for the reader to understand is that bidirectional transformations need not be bijective. A transformation between metamodels M and N given by a relation R is *bijective* if for every model m conforming to M there exists exactly one model n conforming to N such that m and n are related by R, and vice versa (for every model n conforming to N there exists exactly one ...). This is an important special case because there is then no choice about what the transformation must do: given a source model, it must return the unique target model which is correctly related to the source. Ideally, the developer writing a bijective transformation does not have to concern herself with how models should be transformed: it should suffice to specify the relation, which will in fact be a bijective function. (In practice, depending on exactly how the relation is expressed, it might be far from trivial for a tool to extract the functions, however.) Modulo information encoded in the transformation itself, both source and target models contain exactly the same information; they just present it differently. The

classic example in the QVT standard of transformation between a UML class diagram and a database schema is a case where both models contain almost (but not quite) the same information, so it happens not to be a clear illustration of the inadequacy of bijective transformations. More realistically we might express the requirement as the synchronisation of a full UML model, including dynamic diagrams, with a database schema, which makes it obvious that there will be many UML models which might be related to a given schema. More generally, bijective transformations are the exception rather than the rule: the fact that one model contains information not represented in the other is part of the reason for having separate models. The QVT standard [12] is somewhat ambivalent about whether it intends all bidirectional QVT transformations to be bijective. On the one hand, the requirements of MDD clearly imply that it should be possible to write non-bijective transformations (see also [14]): for example, in general, the development of a PSM will involve the addition of information concerning design decisions on a particular platform. On the other hand, it is technically a consequence of statements made in the QVT Relations chapter that all "valid" transformations expressed in that language must be bijective, as we will show below. We take the latter to be a bug in the document, or at least, a restriction which needs to be relaxed for the promise of MDD to be fully realised.

1.2 Related work

In the context of model transformations, almost all formal work on bidirectional transformations is based on graph grammars, especially triple graph grammars (TGGs) as introduced by Schürr (see, for example, [7]). Indeed, the definition of the QVT core language was clearly influenced by TGGs. A master's thesis by Greenyer [4] studies the relationship between QVT (chiefly QVT core) and TGGs, defining a translation from (a simplified subset of) QVT core into a version of TGGs that can be input into a TGG tool. More broadly, the field of model transformations using graph transformation is very active, with several groups working and tools implemented. We mention in particular [8, 13]. Most recently, the paper [2] by Ehrig et al. addresses questions about the circumstances in which a set of TGG rules can indeed be used for forward and backward transformations which are information preserving in a certain technical sense. It is future work to relate our approach to TGGs.

In this context, it may seem foolhardy to write a paper which approaches semantic issues in bidirectional model transformations from first principles. However, there is currently a wide gap between what is desired for the success of MDD and what is known to be soundly supportable by graph transformations; the use of QVT-style bidirectional transformations has not spread fast, despite the early availability of a few tools, partly (we think) because of uncertainty among users over fundamental semantic issues; and moreover, there is a large body of quite different work from which we may hope to gain important insights. Here we give a few pointers.

Benjamin Pierce and colleagues in the Harmony group have explored bidirectional transformations extensively in the context of trees [3], and more recently

in the context of relational databases [1]. In their framework, a *lens*, or bidirectional transformation, is a pair of functions (a "get" function and a "putback" function) which are required to satisfy certain properties to ensure their coherence. They define a number of primitive lenses, and combinators with which to build more complex lenses. Thus, they define a programming language operating on trees in which a program can be read either forwards or backwards. Coherence of the forward and backward readings of the program follows from properties of the primitives and combinators. Their framework is asymmetric, however: their forward transformation is always a function on the source model only, which, in conjunction with their coherence conditions, implies that the target model is always an abstraction of the source model: it contains less information. This has advantages and disadvantages. It is insufficiently flexible to serve as a framework for MDA-style model transformations in general, but the restriction permits certain constructs, especially composition, to work in a way which does not seem to be possible in the more general setting. We shall return to this work later in the paper.

Bidirectional programming languages have been developed in various areas, and a survey can be found in [3]. Notably Lambert Meertens' paper [9] addresses the question of developing "constraint maintainers" for use in user interface design, but his approach is far more general. His maintainers are essentially model transformations which, in terms we shall introduce below, are required to be *correct* and *hippocratic*, but not *undoable*. In [6], Kawanaka and Hosoya develop a bidirectional programming language for XML. In Tokyo, Masato Takeichi and colleagues Shin-Cheng Mu and Zhenjiang Hu have also worked extensively on an algebraic approach to bidirectional programming: see [10, 11, 5].

2 QVT

The OMG's Queries, Views and Transformations (QVT) standard [12] addresses a family of related problems which arise in tool-supported model driven development. Not all information which is modelled is relevant at any one time, so there is a need to be able to abstract out the useful information; and models need to be held in meaningful relationships with one another. Provided that we permit non-bijective transformations (required to support model views), transformations subsume views.

The QVT standard describes three languages for transformations: the Operational, Relational and Core languages. The Relational language is the most relevant here. In the Operational language, someone wishing to specify a bidirectional transformation would have to write a pair of transformations and make them consistent by hand, which we have already said is undesirable. QVT Core is a low level language into which the others can be translated; an example translation from QVT Relational to QVT Core is given in the standard. Since we are concerned with transformations as expressed by users, we will work with QVT Relational.

The issue of whether transformations expressed in QVT Relational are supposed to be bijective is not explicitly discussed, but it seems to be a – possibly unintended – consequence of statements made in [12] that they must be. Specifically, QVT transformations are given a "check then enforce" semantics which means that a transformation must not modify a target model if it is already correctly related to the source model. At the same time, [12] page 18 states:

> In relations, the effect of propagating a change from a source model to a target model is semantically equivalent to executing the entire transformation afresh in the direction of the target model.

This seems to imply that if a transformation is to propagate changes made in a source model m to a target model n, the new target model that results must be independent of the old one: the result of the transformation must depend only on m, since it is equivalent to "executing the entire transformation" on m. In other words given m, there is a unique target model n which must result from executing the transformation. Now suppose that there is also a different model n' which is correctly related to m. Of course, this is quite compatible with the functional interpretation of transformation given in the above quotation: it could happen that even though n' would be a correct target model, n is the one which happens to be produced when the transformation is run on m. In this case, if the transformation is run in a situation where the source model is m and the target model is n', the target model must be transformed into n, even though n' was already correctly related to m. This, however, is exactly what is forbidden by the "check then enforce" semantics: given that n' is already correctly related to m, it must not be modified by running the transformation. If the situation of running a transformation on models which are already correctly related seems too artificial, the reader may prefer to consider a situation in which a target model may be put in correct relation with a source model in two different ways: either by making a tiny change to turn it into one correctly related model, or by making a large change to turn it into a different correctly related model. It will be natural to want the transformation to make the minimal possible changes to ensure relatedness (and, indeed, the text in [12] immediately following the above quotation suggests that this is intended). Interpreting the quoted text literally, though, forbids the transformation to give different results depending on how close the existing target model is to each correctly related target model.

It might be possible to interpret "semantically equivalent" in the above quotation so as to resolve this problem, but this seems forced (since it would require being able to regard models which contained very different information as being "semantically equivalent"). A better solution seems to be to assume that the above quotation is unintentionally restrictive, and disregard it.

3 Semantic issues

In this section we raise a variety of issues which we consider to need further study: they are settled neither by the QVT standard, nor as far as we know by existing related work.

3.1 What exactly it makes sense to write

The QVT Relational language is designed to be easy for someone familiar with related OMG standards such as OCL to learn and use; this has clearly been a higher design goal than ensuring that only safe transformations can be written. There are several places (when and where clauses, among others) where arbitrary OCL expressions are permitted, even though only certain expressions make sense as part of a bidirectional transformation. For example, a transformation may in one direction give an attribute a value using an non-invertible expression.[1] Specifying exactly what language subset is permitted, however, is likely to run quickly into a familiar problem: that any reasonably easy-to-define language subset which is provably safe will also exclude many safe expressions of the full language. It may well be preferable to be permissive, and rely on users not to choose to write things that don't make sense. They will, however, require a clear understanding of what it means for a transformation to "make sense". In Section 4 we propose first steps in this direction and give simple postulates which, we argue, any bidirectional model transformation should obey.

3.2 Determining validity of a transformation

Let us suppose that the reader and the developer accept that model transformations will be written in an expressive, unsafe language, but that the transformations written should obey our proposed postulates (even though this has to be verified on a case-by-case basis, lacking a language in which any transformation is guaranteed to satisfy the postulates). How can developers become confident that their transformations do indeed obey these postulates? Ideally, the language and the postulates should be so clearly understandable that the developers can be confident in their intuition: tool support is no substitute for this kind of clarity. However, it is also desirable that a tool should be able to check, given the text of a transformation and the metamodels to which it applies, that it obeys the postulates. That is, this transformation should be able to be verified statically at the time of writing it, as opposed to failing when it is applied to arguments which expose a problem. Whether or to what extent this can be done is an open question.

A major danger with bidirectional transformations is that one direction of the transformation may be a seldom used but very important "safety net". It will be unfortunate if the user only finds out that their transformation cannot be executed in the less usual direction long after the transformation has been written, in circumstances where the reverse transformation is really needed...

[1] Note that permitting non-bijective transformations does not make this unproblematic: since transformations are to be deterministic, where there are several relationally possible choices of value the language needs to provide a way to specify which should be chosen.

3.3 Composition of relations in QVT: *when* and *where* clauses

Most of this paper takes a high level view of transformations, in which a whole transformation text specifies a relation and a pair of transformational functions. We have not yet considered the details of how simpler relations are combined and built up into transformations in QVT. This is interesting, however, and not least because it gives another justification for considering non-bijective transformations. A QVT relational transformation has an overall structure something like this:

```
transformation ... {
top relation R {
  domain a...
  domain b...
  when {...}
  where {...}
}
top relation S ...
relation ...
relation ...
...
}
```

In order to understand when and where clauses, note that [12] uses two different notions of a relation holding. At the top level, a relation holds of a pair of models – checking will return TRUE – if they are consistent. E.g. if a UML model `m` is consistent with an RDBMS model `s` according to relation `ClassToTable`, we will write $\texttt{ClassToTable}^{+}\texttt{(m,s)}$. The + is intended to distinguish this notion from the following: the consistency between `m` and `s` is *demonstrated* by matching individual classes in `m` to individual tables in `s`: then the class `c` and table `t` (or more formally, the corresponding valid bindings of domain variables in the text of `ClassToTable`) may also be said to be related. We will write `ClassToTable(c,t)`. Note that the relation on models $\texttt{ClassToTable}^{+}$ is a lifted version of the relation on bindings `ClassToTable`: a UML model is related to an RDBMS model by $\texttt{ClassToTable}^{+}$ iff for every class there is a table related to it by `ClassToTable` and vice versa.

The when and where clauses can contain arbitrary OCL, but are typically expected to contain (if anything) statements about relations satisfied by variables of the domain patterns. Thus in fact, the relation `R` holds if for every valid match of the first domain, there exists a valid match of the second domain *such that the where clause holds*. The when clause "specifies the conditions under which the relationship needs to hold". The example used in the standard is the relation `ClassToTable` with domains binding `c:Class` (and hence `p:Package` etc.) and `t:Table` (and hence `s:Schema` etc.), the when clause being `PackageToSchema(p,s)` and the where clause being `AttributeToColumn(c,t)`.

Now, what does this mean in relational terms, and specifically, what is the difference between the when clause and the where clause, both of which ap-

pear at first sight to impose extra conditions on valid matches of bindings, thus forming an intersection of relations? Unfortunately, this is not straightforward to express relationally. Operationally, the idea is that the variables in the when clause "are already bound" "at the time this relation is invoked". Roughly speaking, when a relation `ClassToTable` has domain patterns with variables including `p : Package` and `s : Schema`, and a when clause which states `PackageToSchema(p,s)`, the QVT engine is supposed to have already processed the `PackageToSchema` relation (if not, it will postpone consideration of the `ClassToTable` relation). The matchings calculated for `PackageToSchema` provide bindings for variables `p` and `s` in `ClassToTable`. Evaluation of `ClassToTable` now proceeds, looking for compatible valid bindings of all the other variables.

We have sketched the operational view of what happens in one example, but an open problem is to give a clean compositional account of even the relation (let alone the transformation) defined by a whole QVT transformation. Making this precise would involve a full definition of R^+ taking account of when and where clauses, and an account of the relationship between properties of R and properties of R^+. As an example of the complications introduced by dependencies between relations, suppose that there are two ways of matching pairs of valid bindings (skolem functions) for one relation, one of which leads to a compatible matching for a later-considered relation and one of which does not. If a QVT engine picks "the wrong" matching for the first relation considered, is it permitted to return the result that the models are inconsistent, even though a different choice by the tool would have given a different result? Surely not: but then there is a danger that the tool will need to do global constraint solving or arbitrarily deep backtracking to ensure that it is not missing a solution. Not only is this inefficient, but it will be very hard for the human user to understand. Now, looking at the examples in [12], it seems clear that this kind of problem is not supposed to arise, because when clauses are used in very restricted circumstances. However, it is an open question what can be permitted, and we can expect to encounter the usual problem of balancing expressivity against safety.

For a simple example of "spatial" composition of relations where we *can* lift good properties of simple relations to good properties of a more complex relation, see the next section.

3.4 Sequential composition of transformations

We have discussed the ways in which relations are composed in QVT to make up transformations. A different issue is the sequential composition of whole transformations. We envisage a bidirectional QVT tool which does not retain information between uses: it simply expects to be given a pair of models, a transformation, and a command telling it in which direction to apply the transformation and whether to check or enforce.[2]

[2] If the tool is allowed to retain trace information – the correspondence graph in TGG terms – between executions, the problem becomes more tractable. But this is a severe pragmatic limitation.

We naturally expect to be able to define a transformation to be the sequential composition of two other transformations, and then treat the composition as a first-class transformation in its own right. In this case, the pair of models given to the tool will be the source of the first transformation and the target of the second: the tool will not receive a version of the intermediate model, the one which acts as target of the first transformation and source of the second. In order to define a general way to compose transformations, we need to suppose that we are given transformations R from M to N and S from N to P and show how to construct a composed transformation $T = R; S$, giving the relational and functional parts of the composed relation in terms of the parts of the constituent relations.

We will return to this issue in the next section, after introducing appropriate notation.

4 Requirements for bidirectional model transformations

In this section we discuss bidirectional model transformations which are not necessarily bijective, and discuss under what circumstances these will make sense. We will give postulates which are clearly satisfied by bijective transformations, but also by certain non-bijective transformations.

First let us set some basic notation. We will use capital letters such as R, S, T for the relations which transformations are supposed to ensure. That is, if M and N are metamodels (or sets of models) to be related by a model transformation, the relation $R \subseteq M \times N$ holds of a pair of models – and we write $R(m, n)$ – if and only if the pair of models is deemed to be consistent. Associated with each relation will be the two directional transformations:

$$\overrightarrow{R} : M \times N \longrightarrow N$$

$$\overleftarrow{R} : M \times N \longrightarrow M$$

The idea is that $\overrightarrow{R}$ looks at a pair of models (m, n) and works out how to modify n so as to enforce the relation R: it returns the modified version. Similarly, $\overleftarrow{R}$ propagates changes in the opposite direction.

In practical terms, what we expect is that the programmer writes a single text (or set of diagrams) defining the transformation in the QVT relational language (or indeed, in another appropriate language). This same text can be read in three ways: as a definition of a relation which must hold between pairs of models; as a "forward" transformation which explains how to modify the right-hand model so as to make it relate to the left-hand model; as a "backward" transformation which explains how to modify the left-hand model so as to make it relate to the right-hand model. By slight abuse of notation, we will use capital letters R, S etc. to refer to the whole transformation, including both transformational functions as well as the relation itself, when no confusion can result.

Our notation already incorporates some assumptions, or rather assertions, which need justification.

First, and most importantly, that the behaviour of a transformation should be deterministic, so that modelling it by a mathematical function is appropriate. The same transformation, given the same pair of models, should always return the same proposed modification. This is a strong condition: it proscribes, for example, transformation texts being interpreted differently by different tools. An alternative approach, which we reject, would have been to permit a tool to modify the target model by turning it into *any* model which is related to the source model by the relation encoded in the transformation. There are several good reasons to reject that approach. Most crucially, the model transformation does not take place in isolation but in the presence of the rest of the development process. Even though certain aspects of one model may be irrelevant to users of the other – so that the transformation will deliberately abstract away those aspects – this does not imply that the abstracted away aspects are not important to other users! Usually, it will be unacceptable for a tool to "invent information" in any way, e.g. by making the choice of which related model to choose. The developer needs full control of what the transformation does. Even in rare cases where certain aspects of the transformation's behaviour (say, the choice of name for a newly created model element) might be thought of as unimportant, 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. Experience shows that even when a choice is arbitrary, people find it important that the way the arbitrary choice is made be consistent. One example of this is the finding that, even though the spatial layout of UML diagrams does not (generally) carry semantic information, it is important for UML tools to preserve the information.

Our second assertion is that the behaviour of a transformation may reasonably depend on the current value of the target model which will be replaced, as well as on the source model. This follows from our argument in Section 1 that restricting bidirectional transformations to be bijective is too restrictive. Of course, the fact that we choose a formalism which permits the behaviour of a transformation to depend on both arguments does not force it to do so. In the special case of a bijective transformation, the result of $\overrightarrow{R}$ may be independent of its second argument, and the result of $\overleftarrow{R}$ independent of its first argument. Another important special case is when the transformation in one direction uses only one of its arguments, while the reverse transformation uses both. Pierce et al.'s lenses fall into this category, and we will discuss how they fit into this framework below.

A technical point is that we require transformations to be total, in the sense that $\overrightarrow{R}$ and $\overleftarrow{R}$ are total functions, defined on the whole of $M \times N$. We may want to define, for a metamodel M, a distinguished "content-free" model ϵ_M to be used as a dummy argument e.g. in the case that a target model is created afresh from a source model. Note that since the model containing no model elements might not conform to M, ϵ_M might not literally be empty.

Correctness Our notation is chosen to suggest that the job of $\overrightarrow{R}$ and $\overleftarrow{R}$ is to enforce the relation R, and our first postulates state that they actually do this. We will say that a transformation T is *correct* if

$$\forall m \in M\ \forall n \in N \qquad T(m, \overrightarrow{T}(m,n))$$

$$\forall m \in M\ \forall n \in N \qquad T(\overleftarrow{T}(m,n), n)$$

These postulates clearly have to be satisfied by any QVT-like transformation.

Hippocraticness, or "check-then-enforce" The QVT standard very clearly states that a QVT transformation must not modify either of a pair of models if they are already in the specified relation. That is, even if models n_1 and n_2 are both properly related to m by R, it is not acceptable for $\overrightarrow{R}$, given pair (m, n_1), to return n_2. Formally, we say that a transformation is *hippocratic*[3] if for all $m \in M$ and $n \in N$, we have

$$T(m,n) \implies \overrightarrow{T}(m,n) = n$$

$$T(m,n) \implies \overleftarrow{T}(m,n) = m$$

These postulates imply that if the relation T is *not* bijective, then (at least one of) the transformations must look at both arguments. As a consequence, applying a transformation to a source model in the presence of an existing target model will not in general be equivalent to applying it in the presence of an empty target model.

Undoability Our final pair of postulates is motivated by thinking about the following scenario. The developer, beginning with a consistent pair of models m (the source) and n (the target, perhaps produced by a model transformation), makes a modification to the source model, producing m', and propagates it using the model transformation tool (so that target model n is replaced by $\overrightarrow{T}(m', n)$). Immediately, without making any other changes to either model, our developer realises that the modification was a mistake. She reverts the modified model to the original version m, and propagates the change. The developer reasonably expects that the effect of the modification has been completely undone: just as the modified model has been returned to its original state m, so has the target model been returned to its original state n.

Formally, we will say that transformation T is *undoable* if for all $m, m' \in M$ and $n, n' \in N$, we have

$$T(m,n) \implies \overrightarrow{T}(m, \overrightarrow{T}(m', n)) = n$$

$$T(m,n) \implies \overleftarrow{T}(\overleftarrow{T}(m, n'), n) = m$$

[3] First, do no harm. Hippocrates, 450-355BC

It turns out that this requirement is hard to meet in general, and arguably too strong. However, we find the scenario compelling: a model transformation which did not allow one's changes to be undone in this way would be quite confusing. Therefore, in the present paper, we take it to be essential (we shall shortly show that we can still write non-bijective transformations).

Definition 1. *Let R be a transformation between metamodels M and N, consisting of a relation $R \subseteq M \times N$ and transformation functions $\overrightarrow{R} : M \times N \longrightarrow N$ and $\overleftarrow{R} : M \times N \longrightarrow M$. Then R is a* coherent transformation *if it is correct, hippocratic and undoable.*

4.1 Examples and consequences

Having presented a framework for bidirectional transformations and argued for a set of postulates that they should obey, let us explore the consequences of our choices. (Proofs, all easy, are omitted for space reasons; as are various other small results.) First we state two reassuring trivialities:

Lemma 1. *Let M be any metamodel. Then the trivial transformation, given by:*

- $R(m, n)$ *if and only if* $m = n$
- $\overrightarrow{R}(m, n) = m$
- $\overleftarrow{R}(m, n) = n$

is a coherent transformation.

Lemma 2. *Let M and N be any metamodels. Then the universal transformation, given by:*

- $R(m, n)$ *always*
- $\overrightarrow{R}(m, n) = n$
- $\overleftarrow{R}(m, n) = m$

is a coherent transformation.

Note that the latter lemma already proves that our postulates permit bidirectional transformations which are not bijective. We would of course expect that any bijective transformation is coherent, and so it is:

Lemma 3. *Let M and N be any metamodels. Then any bijective transformation, given by:*

- $R(m, n)$ *if and only if* $n = r(m)$
- $\overrightarrow{R}(m, n) = r(m)$
- $\overleftarrow{R}(m, n) = r^{-1}(n)$

where $r : M \longrightarrow N$ is a bijective function, is a coherent transformation.

The relationship between our framework and that of [3] is close. Note that it is our undoability postulates which prevent more general lenses being coherent.

Lemma 4. *Any* very well behaved *lens l can be regarded as a coherent transformation.*

The reader familiar with [3] may be surprised that our postulates do not include analogues of the GETPUT and PUTGET laws from that work. They are in fact immediate consequences of correctness and hippocraticness:

Lemma 5. *Let M and N be any metamodels, and let R be a correct and hippocratic (but not necessarily undoable) transformation. Then for any $m \in M$, $n \in N$:*

- $\overleftarrow{R}(m, \overrightarrow{R}(m, n)) = m$
- $\overrightarrow{R}(\overleftarrow{R}(m, n), n) = n$

4.2 Composition of metamodels

Let us say that a metamodel M is the *direct product* of metamodels M_1 and M_2 if any model m conforming to M can be written in exactly one way as a pair of a model m_1 conforming to M_1 and a model m_2 conforming to M_2, and conversely, any such pair conforms to M. For example, perhaps M_1 and M_2 comprise disjoint sets of metaclasses, with no relationships or constraints between the two sets. (This is admittedly an artificially constraining scenario: we will discuss relaxations in a moment.)

Now suppose that we have coherent transformations R on $M_1 \times N_1$ and S on $M_2 \times N_2$. We can construct a transformation which we will call $R \oplus S$ on $M \times N$ pointwise as follows:

- $(R \oplus S)(m_1 \oplus m_2, n_1 \oplus n_2)$ if and only if $R(m_1, n_1)$ and $S(m_2, n_2)$
- $\overrightarrow{(R \oplus S)}(m_1 \oplus m_2, n_1 \oplus n_2) = (\overrightarrow{R}(m_1, n_1)) \oplus (\overrightarrow{S}(m_2, n_2))$
- $\overleftarrow{(R \oplus S)}(m_1 \oplus m_2, n_1 \oplus n_2) = (\overleftarrow{R}(m_1, n_1)) \oplus (\overleftarrow{S}(m_2, n_2))$

Then

Lemma 6. *If R and S are coherent transformations, $R \oplus S$ is also a coherent transformation.*

The proof is a straight application of the definitions. This captures the intuition that transformations on parts of models which are completely independent ought to be able to be combined without difficulty. One would expect to be able to extend this result to cover carefully-defined simple dependencies between the metamodel parts, perhaps sufficient to justify, for example, applying a transformation defined only for class diagrams to a complete UML model, rolling the resulting changes to the class diagram through to the rest of the model. Even here, though, the issues are not entirely trivial.

4.3 Sequential composition revisited

The relation part of the sequential composition of transformations must be given by the usual mathematical composition of relations: $(R; S)(m, p)$ if and only if *there exists some* n such that $R(m, n)$ and $S(n, p)$. Mathematically this is a fine definition, but we already see the core of the problem: a tool has no obvious way to find a relevant n. What about the associated transformations? For example, $\overrightarrow{T}$ may be given models m and p such that there does not exist any n such that $R(m, n)$ and $S(n, p)$. It is required to calculate an update of p; that is, to find a new model p' such that such an intermediate model does exist, and in general the choice of intermediate model will depend on both m and p. However, $\overrightarrow{R}$ "does not understand" p, etc., so there does not appear to be any way to do this in general.

We may consider two special cases in which it is possible to define composition of transformations.

1. If R and S are bijective transformations, then the intermediate model is unique, and is found by applying $\overrightarrow{R}$ just to the first argument m. Composition of transformations in this case is just the usual composition of invertible functions.
2. More interestingly, the Harmony group considers transformations in which $\overrightarrow{R}$ is a function of the source model only, even though $\overleftarrow{R}$ still uses both source and target model. Here $\overrightarrow{R; S}$ must clearly be defined to be $\overrightarrow{R}; \overrightarrow{S}$, and we can define $\overleftarrow{R; S}$ using a trick: use the function $\overrightarrow{R}$ to bring the source model forward into the middle in order to use it to push the changes back. Formally (and translating into our notation)

$$\overleftarrow{R; S}(m, n) = \overleftarrow{R}(m, \overleftarrow{S}(\overrightarrow{R}(m), n))$$

5 Conclusion

We have explored some fundamental issues which arise when we consider relationally defined transformations between models which are bidirectional and not necessarily bijective. We have motivated our work from the current QVT standard, and some of the issues we raise are specific to it, but most are more general. We have suggested a framework and a set of postulates which ensure that bidirectional transformations will behave reasonably for some definition of "reasonable", and explored some consequences of our choice. Future work includes relating our framework to triple graph grammars, and further exploration of the relation with bidirectional programming.

Acknowledgements The author would like to thank Reiko Heckel, Conrad Hughes, Gabriele Taentzer and especially Benjamin Pierce for helpful discussions.

References

1. Aaron Bohannon, Jeffrey A. Vaughan, and Benjamin C. Pierce. Relational lenses: A language for updateable views. In *Principles of Database Systems (PODS)*, 2006. Extended version available as University of Pennsylvania technical report MS-CIS-05-27.
2. Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information preserving bidirectional model transformations. In *In proceedings of Fundamental Approaches to Software Engineering (FASE 2007)*, number 4422 in LNCS, pages 72–86. Springer, March/April 2007.
3. J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *ACM Transactions on Programming Languages and Systems*, 2007. to appear: preprint available from http://www.cis.upenn.edu/ bcpierce/papers/index.shtml.
4. Joel Greenyer. A study of technologies for model transformation: Reconciling TGGs with QVT. Master's thesis, University of Paderborn, Department of Computer Science, Paderborn, Germany, July 2006.
5. Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. In *In Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation (PEPM'04)*, pages 178–189, 2004.
6. Shinya Kawanaka and Haruo Hosoya. biXid: a bidirectional transformation language for XML. In *In Proceedings of the International Conference on Functional Programming, ICFP'06*, pages 201–214, 2006.
7. A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars - A Survey. In R. Heckel, editor, *Proceedings of the SegraVis School on Foundations of Visual Modelling Techniques*, volume 148 of *Electronic Notes in Theoretical Computer Science*, pages 113–150, Amsterdam, 2006. Elsevier Science Publ.
8. Alexander Königs. Model transformation with triple graph grammars. In *In proceedings, Workshop on Model Transformations in Practice*, September 2005.
9. Lambert Meertens. Designing constraint maintainers for user interaction. Unpublished manuscript, available from http://www.kestrel.edu/home/people/meertens/, June 1998.
10. Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. An algebraic approach to bidirectional updating. In *In Proceedings of Programming Languages and Systems: Second Asian Symposium, APLAS'04*, pages 2–20, 2004.
11. Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. An injective language for reversible computation. In *In Proceedings of Mathematics of Program Construction (MPC'04)*, pages 289–313, 2004.
12. OMG. MOF2.0 query/view/transformation (QVT) adopted specification. OMG document ptc/05-11-01, 2005. available from www.omg.org.
13. Gabriele Taentzer, Karsten Ehrig, Esther Guerra, Juan de Lara, Laszlo Lengyel, Tihamer Levendovsky, Ulrike Prange, Daniel Varro, and Szilvia Varro-Gyapay. Model transformation by graph transformation: A comparative study. In *Workshop on Model Transformations in Practice*, September 2005.
14. Steven Witkop. MDA users' requirements for QVT transformations. OMG document 05-02-04, 2005. available from www.omg.org.