A Landscape of Bidirectional Model Transformations

Perdita Stevens

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
Fax: +44 131 667 7209
perdita@inf.ed.ac.uk

Abstract. Model transformations are a key element in the OMG's Model Driven Development agenda. They did not begin here: the fundamental idea of transforming, automatically, one model into another is at least as old as the computer, provided that we take a sufficiently broad view of what a model is. In many contexts, people have encountered the need for *bidirectional* transformations. In this survey paper we discuss the various notions of bidirectional transformation, and their motivation from the needs of software engineering. We discuss the state of the art in work targeted specifically at the OMG's MDD initiative, and also, briefly, related work from other communities. We point out some areas which are so far relatively under-researched, and propose research topics for the future.

Keywords: bidirectional model transformation, QVT, graph transformation, triple graph grammar, bidirectional programming language.

1 Introduction

Bidirectional model transformations are both new and old. New in that they have recently come to prominence in the context of the Object Management Group's Model Driven Architecture initiative. Old in the sense that the problems which arise in considering them turn out to be similar to problems that have been studied in other contexts for decades.

In this paper, we explore the landscape of bidirectional model transformations as it appears to the author at the time of writing, autumn 2007 to spring 2008. We attempt to summarise the perceived need for bidirectional model transformations, drawing out the various different assumptions that may be made about the environment in which the transformation operates, since this has important effects on the technology and underlying theory available. We discuss the state of the art in model transformations in the OMG sense, and then move on to survey related work in other fields. Finally, we mention some areas which have been relatively under-studied to date, and suggest some directions for further research.

2 Background: Why Bidirectional Model Transformations?

A model, for purposes of this paper, is any artefact relating to a software system. For example, its code or any part of it; UML models for it, or for a product-line of which it is a member; its database schema; its test set; its configuration management log; its formal specification, if any. We will be mostly concerned with formal artefacts and will therefore not usually consider, for example, the natural language document that describes the systems architecture, or the physical computer on which the system will run, to be models; however, it is surprisingly difficult to draw a hard-and-fast line between formal and informal artefacts.

Model transformations aim to make software development and maintenance more efficient by automating routine aspects of the process. For example, a common scenario is that two models have to be consistent in some sense, and that it is possible to codify the process of restoring consistency when one model is changed. In such a case, it is helpful if this notion of consistency and restoration process can be captured in a model transformation. In the simplest case, only one of the two models (the *source*) is ever modified by humans: the other (the *target*) is simply generated by a tool. The tool is deterministic, or at least, if it is possible for it to produce several different targets from the same source, the various targets are seen as having equal value, so that the difference between them is immaterial. In such a case, in practice we will never check consistency: if there is any doubt that the target was produced from the source, the transformation will be re-run. The old target model is discarded and replaced by the newly-generated version. For example, this is typically the case when a program in a high-level language is compiled. If the sets of models are M and N, a unidirectional transformation of this kind can be modelled as $f : M \longrightarrow N$; in the simplest case, the consistency relation will be simply that m and n are consistent iff $n = f(m)$.

More interesting, more difficult and arguably more typical is the situation where both models may be modified by humans. Examples include:

1. The classic MDA situation: a platform independent model (PIM) is transformed into a platform specific model (PSM).
2. The database view problem, in which a user is presented with a view of a selected subset of the data from the database, and may modify the view: at the same time, the full database may change in the usual way.
3. Many kinds of integration between systems or parts of systems, which are modelled separately but must be consistent: for example, a database schema must be kept consistent with the application that uses it.

Even if the consistency restoration process has to be manual – for example, because there may be several different ways to restore consistency, with different values, and nobody can formalise rules which capture which way is best – it may still be worthwhile to have an automatic check of whether the models are consistent.

A *bidirectional model transformation* is some way of specifying algorithmically how consistency should be restored, which will (at least under some circumstances) be able to modify either of the two models. Within this broad definition, there are many variants which we shall explore.

Even when transformation technologies are first thought of as unidirectional, it often turns out that bidirectionality is required. In [32], it is interesting to note that the ability to write bidirectional transformations appears high in the list of users' priorities, even though the original call for proposals for the OMG's language for expressing Queries, Views and Transformations, QVT [28] listed bidirectionality as an optional feature.

2.1 Related Work

Consistency management has as mentioned a long history, which we will not go into here. In the field of models, an interesting early paper focusing on pragmatic issues that arise in round-trip engineering – connecting a model with source code – is Sendall and Küster's position paper [29]. Czarnecki and Helsen in [8] present a survey of model transformation techniques with a particular emphasis on rule-based approaches such as those based on graph transformations. They mention directionality, but do not focus on it. Similarly Mens and Van Gorp in [24] discuss the wider field of model transformations, not necessarily bidirectional.

The present author in [30] discussed bidirectional transformations in the specific context of the QVT Relations (hereinafter QVT-R) language. The paper proposed a framework for bidirectional transformations in which a bidirectional transformation is given by three elements, flexible enough to describe any of the approaches considered here. We write $R(m,n)$ if and only if the pair of models is deemed to be consistent. Associated with each relation R 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. The paper proposed postulates to ensure that the transformation should be coherent.

2.2 Terminology

In a truly symmetric situation, there is an arbitrary choice about which model to call the "source" and which the "target": nevertheless, the terminology is widely used and useful, and there is usually some asymmetry in the situation. Generally, the "source" model is the one which is created first, and it is more probable that a change to the source model will necessitate a change to the target model than the other way round.

Note that this paper does not attempt to discuss extra problems that may arise when checking or restoring consistency between more than two models – multidirectional transformations.

3 Important Differences in Circumstances

Discussions of bidirectional transformations often turn out to hinge on what assumptions are made about the situation: a solution which seems good with one set of assumptions can be infeasible with another. In this section, we draw out some of the most important differences in circumstances of use of a transformation. These differences are among the chosen design assumptions of the developers of a bidirectional transformation approach; looked at from the point of view of users who need to choose an appropriate approach for their circumstances, they are factors that may influence that choice. Our aim here is not to classify existing approaches: indeed, in this section we will discuss several issues where all the approaches we will later cover are in accord. Such issues are not useful for distinguishing between existing approaches, but they may be useful in pointing out lacunae in the current research. This is our main aim.

In later sections, we will discuss approaches to bidirectional transformations, trying to make clear how they fit into the framework of differences presented in this section. However, we do not claim to have exhaustively addressed every issue for every tool. Where there is a majority approach, we say so in this section, and point out exceptions later.

3.1 Is There an Explicit Notion of Two Models Being Consistent?

Situations where bidirectional transformations may be applied always involve at least an informal notion of consistency: two models are consistent if it is acceptable to all their stakeholders to proceed with these models, without modifying either. Since we have said that the job of the bidirectional transformation is to restore consistency, we may alternatively say that two models are consistent if no harm would result from not running the transformation. Note that this is not quite the same as to say that running the transformation would have no effect: some transformation approaches may modify a pair of models, even if they are already consistent.

Typically, bidirectional transformation languages described as "relational", such as QVT-R and BOTL (to be discussed), make their notion of consistency explicit: part of what they do is to define a consistency relation on two sets of models. Many other presentations of bidirectional transformations do not make their notion of consistency explicit. In such a case, any situation which may arise from a (successful) application of the transformation may be considered consistent by definition. However, it can be hard to characterise this other than by describing exactly what the transformation tool will do.

3.2 Does One Expression Represent Both Transformation Directions?

Superficially, the easiest way to write a bidirectional transformation is to write it (in text or in diagrams) as a pair of functions, one in either direction, maybe also giving an explicit consistency relation. This avoids the need to write special purpose languages for bidirectional transformations, enabling instead the reuse of established programming languages. However, even if we make explicit constraints on the two functions which suffice to make the different expressions coherent, we have an obvious danger: that the coherence of the different expressions will not be maintained as time passes.

Notice that the concern here is how actual transformations are written down. There is no objection to modelling a transformation as a pair of functions, if both functions are represented by one expression (textual or graphical).

Perhaps, if a framework existed in which it were possible to write the directions of a transformation separately and then check, easily, that they were coherent, we might be able to have the best of both worlds. However, no such framework exists today.

Since this paper is focused on work which aims specifically at addressing bidirectional transformations, it is unsurprising that all the approaches to be discussed involve writing a single text (or equivalently, a single set of diagrams).

3.3 Is the Transformation Bijective, Surjective or Neither?

This is probably the most important assumption to clarify when looking at a particular tool or technology: however, this is not always easy if the approach does not have an explicit notion of consistency (because one may have to study the syntax of the language to understand which implicit consistency relations can be defined).

Given a consistency relation on the pairs of models, we call a transformation bijective if the consistency relation is a bijection: that is, for any model (source, rsp. target) there exists a unique corresponding model (target, rsp. source) which is consistent with it. This is a very strong condition: in effect, it says that the two models contain identical information, presented differently. In particular, it is impossible to define a bijective bidirectional transformation between two sets of models which have different cardinalities.

If the reader has in mind an approach whose notion of consistency is implicit, this may not be quite obvious. Consider, for example, a transformation which defines (in any way) a function

$$f : M \longrightarrow N$$

with the implicit understanding that m and n are consistent iff $n = f(m)$. This consistency relation is a bijective relation if and only if f is a bijective function, and in that case there is an inverse function

$$f^{-1} : N \longrightarrow M$$

satisfying the usual identities. This situation cannot arise if, to give a trivial example, completely defining $m \in M$ requires three boolean values while completely defining $n \in N$ requires three integer values.[1]

Thus, an approach that permits only bijective transformations is in most practical situations much too restrictive. A more realistic, but still quite restrictive, condition is that one of the models should be a strict abstraction of the other: that is, one model contains a superset of the information contained in the other.

3.4 Must the Source Model Be Modified When the Target Changes?

This issue does not strictly concern bidirectional transformations, but it does concern transformations which do not fit into the unidirectional transformation framework ($f : M \longrightarrow N$) initially presented.

Suppose we have a source and target model which are consistent, perhaps because the target was produced from the source by compilation. Suppose further that changes may be made to the target model, but they are small in the sense that they will not cause the target to become inconsistent with the source. Two examples are:

1. Optimising the code produced by a compiler (using a suitably restrictive class of optimisations)
2. Taking skeleton code generated from a UML class diagram, and adding method bodies.

(Notice that the allowable changes to the target depend crucially on the notion of consistency chosen.)

In this restricted case, changes to the target model never require changes to the source model. The reason why this does not fit into the straightforward unidirectional transformation framework presented above is that it is not acceptable to discard changes to the target, when the source model changes and the target needs to be updated to reflect those changes. This kind of transformation might be represented as a function

$$f : M \times N \longrightarrow N$$

assuming that a default value from N, representing "no interesting information" is available for use in the initial generation.

All the approaches discussed here assume that the source model may have to be modified when the target changes.

3.5 Must the Transformation Be Fully Automatic, or Interactive?

If there may be a choice about how to restore consistency, there are two options: either the transformation programmer must specify the choice to be made, or

[1] Of course, which model sets actually have the same cardinality depends crucially on whether we define our models using machine integers or mathematical integers, etc.; but either way, not all model sets have the same cardinality.

the tool must interact with a user. The ultimate interactive bidirectional transformation would consist simply of a consistency checker: the tool would report inconsistencies and it would be up to the user to restore consistency. Intermediate scenarios, in which the programmer specifies how most kinds of inconsistency can be dealt with, but exceptional cases are referred to the user, might be imagined. Another variant, [7], is discussed later.

Any approach which involves a tool behaving non-deterministically has an obvious interactive counterpart in which the user resolves the non-determinism; we will mention one such case. However, none of the work discussed here explicitly addresses interaction with the user.

3.6 Is the Application of the Transformation under User Control?

A slightly different question is whether the user of the transformation tool controls when consistency is restored, or whether this is done automatically without user involvement. Practically, applying the transformation automatically entails that the transformation should be fully automatic, but not vice versa. There are advantages to automatically applied transformations (compare automatic recompilation of changed source files in Eclipse and similar development environments, or the demand for "pushed" changes to data e.g. web pages): the user does not have to remember to apply the transformation, and the danger of work being wasted because it is done on an outdated model is reduced. However, the fact that users' actions have effects which are invisible to them may also be confusing, e.g. if the transformation is not undoable in the sense of [30]. Intermediate situations such as transformations being applied every night, or on version commit, may sometimes be good compromises.

3.7 Is There a Shared Tool?

If the people modifying the two models use the same model transformation tool, then the tool may perhaps keep and make use of information other than the models themselves. For example, it may keep what is called "trace" information in QVT or "correspondence" information in triple graph grammars: a persistent record of which elements of one model are related to a given element in the other. This can simplify the programming of model transformations, in that provided the notion of corresponding elements can be somehow established, it can be used throughout the transformation definition. If this extra information were itself extractable in a standard format which could be read by several tools, then of course the same could be done even without a literally shared tool, at the expense of giving the user another artefact to manage.

3.8 Is It Permissible to Cache Extra Information in the Models?

As discussed above, there may be information in each model which is not contained in the other. One approach is to pick one of the models and modify it so that it contains all of the information from both models, in a form which is not evident to the user of the model. For example, if we consider a transformation between a UML model and the code that implements it, the method bodies may be hidden in the model, and/or the diagrammatic layout information may be encoded in comments in the generated source files.

By this means, the bidirectional transformation can be made surjective, or even bijective, while leaving the models looking the same to their users. In effect, we are adding extra transformations: instead of a general bidirectional transformation

$$M \longleftrightarrow N$$

we have

$$M \hookrightarrow M' \longleftrightarrow N' \hookleftarrow N$$

where the outer transformations are implicit, being invisible to the user, and it can be arranged that the inner transformation is bijective.

This is a very useful and pragmatic approach widely used in, for example, generation of code from UML models. It carries the same kind of problems as any other kind of non-user-visible transformation, however. There is a possibility that the user, not understanding the extra information, may accidentally modify it, for example. Moreover, this approach only works if the transformation tool has complete control over the models. If a different tool is also being used to modify one of the models, information may be lost.

3.9 What Must the Scope of the Model Sets Be?

Much of the work to be discussed works with XML; some with relational database schemas; some with MOF metamodels. Some problems are more tractable if the language of models can be restricted. For example, [2] discusses round-trip engineering of models in the context of a framework-specific language. If the models are trees, e.g. XML documents, are idrefs and similar ways to turn a tree into a graph permitted? If lists occur, is order important? If the models are graphs, are attributes permitted, and with what restrictions?

4 The QVT Relations Language and Tools

QVT-R is the relational language which forms part of the OMG's Queries, Views and Transformations standard [28]. A single text describes the consistency relation and also the transformations in both directions. [30] discusses the use of the language for bidirectional transformations, pointing out in particular that there is some ambiguity about whether the language is supposed to be able to describe transformations which are not bijective.

Despite the heavy emphasis placed on model transformation by the OMG's model-driven development strategy, and the clear importance of bidirectionality to users, tool support for bidirectional transformations expressed in QVT-R remains limited. However, IKV++'s Medini QVT[2] incorporates an open-source

[2] `http://projects.ikv.de/qvt`, last accessed March 13th 2008.

QVT-R engine; TCS's ModelMorf tool[3] is also available, but is still in "pre beta" release, with no clear sign of a forthcoming true release. QVT-R is not yet supported in the Eclipse M2M project, though such support is planned.

5 Approaches Using MOF but Outside QVT

ATL, the ATLAS Transformation Language, is a widely used transformation language often described as "QVT-like", which has good tool support. (The term ATL gets used both for the entire architecture and for the middle abstraction layer.). It is a hybrid language, encouraging declarative programming of transformations but with imperative features available. Its most abstract layer, the ATLAS Model Weaving language (AMW) [10] [4] allows the specification of a consistency relation between sets of models, which as we have discussed is an important aspect of bidirectional transformations. AMW does not, however, provide general capabilities for the programmer to choose how inconsistencies should be resolved; in ATL, a bidirectional transformation must be written as a pair of unidirectional transformations [17]. In passing, let us remark that while [10] distinguishes model weaving from model transformation, the terminological distinction of that paper does not seem to be in widespread use: both concepts, as described there, are within "model transformation" as covered in this paper, and in fact the term "model weaving" is used in other senses by other authors.

MOFLON [1] provides, among other things, the ability to specify transformations using triple graph grammars (discussed below).

BOTL [5], the Basic (or elsewhere, Bi-directional) object-oriented transformation language, builds from first principles a relational approach to transformation of models conforming to metamodels in a simple sublanguage of MOF. Although it discusses the point that relations may not be bijective, when considering transformations it only addresses bijective relations: the language does not provide the means to specify how consistency should be restored if there is a choice.

6 XML-Based Options

A number of approaches to bidirectional transformations between XML documents (or between an XML document and another kind of model) have arisen in the programming languages community: programming languages, possibly with sophisticated type systems, have been developed for writing single programs which can be read as forward or as backward transformations. Unlike the graph transformation approach, a program is conceived as a whole, with a defined order of execution and some degree of statically checkable correctness.

Of these, biXid [18] is the most similar to QVT-R and graph grammar work, as it adopts a relational approach to bidirectional transformations between pairs of

[3] See http://www.tcs-trddc.com/ModelMorf/index.htm, last accessed March 13th 2008.

[4] See also http://www.eclipse.org/gmt/amw/, last accessed March 13th 2008.

XML languages. It allows non-bijective relations (ambiguity, in the terminology of the paper) and discusses the implications of this for the transformation engine; however, a consistent model is chosen non-deterministically from among the possibilities. There is deliberately no way for the programmer to specify how consistency should be restored when there is a choice. This makes sense in the context addressed, which is where different formats for the same information must be synchronised: the non-bijectivity is expected to come from the existence of different but semantically equivalent orderings of the same information, not from the presence of different information which must be retained. It also makes a point of the design decision that a variable on one side of a relational rule may occur several times on the other side (non-linearity), which is important for permitting the restructuring of information.

The Harmony project [3,12] defines a language called Boomerang (building on earlier work on a language called Focal) in which bidirectional programs – termed lenses – are built up from a deliberately limited set of primitive lenses and combinators. Compositionality is thus a major concern of the approach, in contrast to the relational approaches. Transformations are always between a “concrete” set of models and a strict abstraction of this, the “abstract” set of models which contains strictly less information: hence, there is an explicit notion of consistency in which two models are consistent exactly when one is an abstraction of the other. The transformation programmer has control of how consistency should be restored where there is a choice (which only happens when the abstract model has been modified and the concrete model needs to be changed).

Hu, Mu and Takeichi [16,25,26] define a language called Inv, a language defined in “point free” style (that is, without variable names: this has obvious advantages for bidirectional languages, as also argued by [27], but can be a readability barrier) in which only injective functions can be defined. In order to capture useful behaviour, the codomain is enriched with a history part, thus making an arbitrary function injective. In model transformation terms, this approach modifies the metamodel in order to be able to define transformations. The language is targeted at the authors’ Programmable Structured Document project: [16] develops an editor for structured documents, in which the user edits a view of a full document, and these edits induce edits on the transformation between the full document and the view. This is different from the more usual bidirectional transformation scenario in which the two models are disconnected while they are being edited: that is, the work assumes a shared tool, although note that it does *not* assume that the transformation is automatically applied. A view in this sense may duplicate information, retaining dependencies between the copies – the transformation may be non-linear – but it is still the case that a document uniquely determines a view, that is, the transformation is surjective.

XRound [6] is a template-based language for bidirectional transformations. Template-based transformation languages have been widely used for unidirectional transformations such as code generation from models. In that application, the transformation gives a boiler-plate piece of code containing metavariables. The metavariables are replaced by values obtained by interrogating the input model, to give the output code. The novelty in XRound is to use a similar technique bidirectionally by using unification between variables of the target application (e.g., from the output code) and the results of queries on the input model. Thus XRound provides a bidirectional, but asymmetric, language of transformations particularly adapted to cases where an XML document must be synchronised with a simpler structure associated with another tool. The original application was security analysis of models presented as XMI files.

Finally, XSugar [4], which, being aimed at transforming between two syntaxes for the same language, naturally insists that transformations should be bijective.

It is tempting to think that the work on transformations of XML documents should be immediately applicable to models in MOF-based languages, since these can be represented by XMI files. However, there are significant obstacles not yet overcome. Although XML files represent trees, models typically make essential use of `xmiids` or similar mechanisms to turn the trees into general graphs. None of the existing XML transformation languages work well in this context ([16], for example, explicitly forbids `IDRef` use, though it is not clear that it forbids the use of other information for the same purpose.)

7 Graph Transformations

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, [20]). Two graphs represent the source and target models, and a third graph, the correspondence graph, records information about the matches between nodes in the source and target. Thus TGGs are an approach which relies on developers of the source and target model using a shared tool which can retain this information, or at least on its being retained somehow, possibly by modification of one of the two metamodels. Each rule has a top, or precondition, part which specifies when the rule applies: that is, what triples consisting of a subgraph of each of the three graphs should be examined. The bottom part of the rule then specifies consistency by describing what else should be true of each of the three graphs in such a case. The transformation programmer controls how consistency should be restored by marking in the bottom part of the rule which nodes should be created if no such node exists in its graph. Thus, it is possible to use TGGs to specify bidirectional transformations with non-bijective consistency relations. However, extending this to programming model transformations introduces new issues, such as how to describe how to restore any consistency relations that are specified between attributes that are not modelled as part of the graphs.

For a discussion of the use of TGGs for model transformation, see [14]. Indeed, the definition of the QVT core language was clearly influenced by TGGs. Greenyer and Kindler have given an informative comparison between QVT core and triple graph grammars in [13]. In the process of developing a translation of (a simplified version of) QVT core to a variant of TGGs that can be input into a TGG tool, they point out close similarities between the formalisms and discuss some semantic issues relating to bidirectionality.

The main tool support for TGGs is FUJABA [5], which provided the foundation for MOFLON mentioned above. FUJABA (like other tools) provides the means for rules to be given different *priorities* in order to resolve conflicts where more than one rule applies. This can improve the performance of a transformation engine, but it can also be another means for the programmer to specify how consistency is restored when there is a choice. Anecdotally, it can be hard to understand the implications of choices expressed in this way and hence to maintain the transformation.

More broadly, the field of model transformations using graph transformation is very active, with several groups working and tools implemented. We mention in particular [21,31]. Most recently, the paper [9] 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.

7.1 Miscellaneous

Meertens [23] writes about bidirectional transformations – “constraint maintainers” – in a very basic and general setting of sets and relations. His maintainers are given as three components, a relation and a transformation in each direction. He discusses the conditions which relate them, with particular concern for minimising the “edit distance” involved in restoring consistency.

Finally, [7] is unusual in exploring the effects of using interactivity of a tool to loosen the constraints imposed by the metamodels. If a target model is generated from a source by a model transformation, and then manually modified, possibly in such a way that it no longer conforms to the target metamodel, how should consistency be restored between this modified target model and the original source? The authors propose using Answer Set Programming to produce a set of possible source models which, on application of the original forward transformation, give target models close to the manually modified one. The user then chooses from among these “approximations”.

8 Research Directions

At the time of writing, autumn 2007, there is still a wide gap between the vision of model transformations as first-class artefacts in the software engineering of systems, and the reality of the techniques and tools available. In this relatively immature field there is obviously a need for further work in many directions. In this section, we focus on some areas which seem to have been relatively under studied, but which will be important in practice.

8.1 Compositionality

To get beyond toy examples, we need clean mechanisms to compose model transformations in (at least) two senses. We need good semantic understanding of the

[5] `http://www.fujaba.de`, last accessed March 13th 2008.

consequences of composing transformations, and also good engineering understanding and tool support to make it practical to do so, even when the transformations originate from different groups. Both of these senses sound fairly unproblematic, but in fact, problems can arise: see [30] for more discussion.

Spatial composition. If two systems are composed of parts (such as components or subsystems) which themselves can be acted on by model transformations, we will need to be able to compose those part transformations to give a transformation of the whole systems. We will need to be able to understand the effect of the composed transformation by understanding the parts. Ideally, this would apply even if the system parts were not themselves encapsulated parts of the systems: for example, given transformations of aspects of systems, we would like to be able to construct transformations of the woven systems.

Sequential composition. Given unidirectional transformations $f : A \longrightarrow B$ and $g : B \longrightarrow C$, we can obviously compose them to yield $gf : A \longrightarrow C$. This extends to bidirectional transformations if they are bijective. In the general case, however, composition does not work because of the need to "guess" matching elements in the elided middle model space. Part of the attraction of allowing bidirectional transformations to relate models to their strict abstractions is that it permits a reasonable notion of composition: see [30], and also [23] for discussion.

8.2 Specification

Specifications of programs are (ideally) useful in a variety of ways. Because they focus on what the program must do, not how, they can be written before the program, and can guide the programmer in writing the program that is required. Depending on the kind of specification, we may be able to verify the program against the specification, and/or we may be able to check at runtime than the specification is not violated by a particular run of the program. If we do find a discrepancy between the specification and the program, this may as easily indicate a defect in the specification as in the program; but either way, understanding is improved. The specification can also be used by potential reusers of the program, and for various other purposes. The most useful specifications are precise, but simpler and easier to understand than the program they specify.

What kinds of specification of model transformations may prove useful? How can we write specifications of model transformations which are simpler than the transformations themselves? One pragmatic approach would be to have a precise formal specification of the intended consistency relation, together with natural language description of how to choose between several consistent models.

8.3 Verification, Validation and Testing

To date, most work on verification or validation of model transformations turns out to be concerned not with verification or validation in the usual software engineering sense, of ensuring that the transformation conforms to its specification and correctly expresses the user's requirements, but in the special sense of making sure that the transformation is sane. For example, [22] focuses on ensuring that a graph transformation is terminating and confluent; that is, well-defined. This is obviously crucial, but after this is achieved, we still have to address the "are we building the right transformation?" sense of validation and the "are we building the transformation right?" sense of verification.

To validate, or to test, an ordinary function, we will normally run it on various inputs. A difficulty in importing this to model transformations is the relative difficulty of constructing or finding suitable input models, which will tend to limit the amount of testing or validation done. Model transformations are expected to be written in a wide range of business contexts. If the model transformation itself is regarded as a product having value, it may be seen as legitimate to invest effort in validating it. If, however, a model transformation is developed for use within a single organisation, perhaps initially on a small range of models, this less likely. Compounding this problem is the complexity of the models to which transformations are likely to be applied, and the difficulty of telling at a glance when a transformation is erroneous.

Given a precise specification of a model transformation, and the definition of the transformation in a semantically well-defined form, we may expect to be able to apply the body of work on formal verification of programs, and perhaps more usefully of concurrent systems. This brief statement, though, doubtless hides a need for hundreds of papers to work out the details.

In the field of testing, too, we will have to answer again a host of theoretical and engineering questions. How can test suites be generated (semi-)automatically? What are appropriate coverage criteria, and can they be automatically checked? Can mutation testing be helpful? etc.

8.4 Debugging

Once a test or a verification fails, the model transformation must be debugged: that is, the user must be helped to localise, understand and fix the problem. This may be done "live", in the tool that is applying the transformation, or "forensically", based on records of the transformation. In the latter field, [15] is interesting early work, done in the context of Tefkat transformations with link (traceability) information available. Much more remains to be done, however. We may note that even the debugging facilities available for models themselves are not yet very sophisticated, compared with what is routinely used for programs.

8.5 Maintenance

To date, we lack serious experience of what issues may arise in the maintenance of model transformations. Experience with maintenance of other software artefacts suggests that easily usable specifications will be useful, but that they will, in any case, get out of date; that readability of the representation will be crucial; and that a notion of refactorings of model transformations will be needed to prevent architectural degradation of large transformations. We may expect in the long run to have to define model transformation languages with clear modularity permitting well-defined interfaces that can hide information; this returns us to the issue of compositionality. What the relationship should be between modularity of models and modularity of model transformations also remains to be seen. A concrete fear is that pattern-based identification of areas of a model where a transformation is relevant may possibly turn out to be particularly fragile. [19] describes interesting early work in this field, focusing on structuring and packaging bidirectional model transformations, specifically TGGs.

8.6 Tolerating Inconsistency

Anyone using bidirectional transformations asks: to what extent can inconsistency be tolerated? If not at all, then we need frameworks in which edits are applied simultaneously to both models, even though the user doing the editing sees only part of the effect of their edit. In most of the work we have discussed, the assumption is that inconsistency can be tolerated temporarily – it is acceptable for one, or even both, models to be edited independently – but that consistency must eventually be restored by the application of a model transformation. The other end of the spectrum is to accept that the two models will be inconsistent and work on pragmatic means to tolerate the inconsistency. But is this different from using a different, weaker notion of consistency? The body of work exploring this territory includes for example [11]: it might prove fruitful to explore applications to model driven development.

9 Conclusions

We have surveyed the field of bidirectional model transformations, as it appears at the beginning of 2008. We have pointed out some of the assumptions between which the developers of model transformation approaches must choose, and we have discussed some of the main existing tools. Finally, we have pointed out some areas where further work is needed.

What general conclusion can we draw about the future of bidirectional model transformations? Do they, indeed, have a long-term future, or will the languages developed for writing them die without being replaced? This paper has pointed out a number of areas – support for interactivity, for debugging, for verification, validation and testing, for maintenance, among others – that are only beginning to be addressed. In the author's opinion, realisation of the full potential of bidirectional transformations depends on progress in these areas.

Acknowledgements. The author would like to thank the anonymous reviewers for their comments, and Joel Greenyer, Reiko Heckel, Conrad Hughes, Gabriele Taentzer and especially Benjamin Pierce for helpful discussions. Any errors are the author's own.

References

1. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066. pp. 361–375. Springer, Heidelberg (2006)
2. Antkiewicz, M., Czarnecki, K.: Framework-specific modeling languages with round-trip engineering. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 692–706. Springer, Heidelberg (2006)
3. Bohannon, A., Foster, J.N., Pierce, B.C., Pilkiewicz, A., Schmitt, A.: Boomerang: Resourceful lenses for string data. In: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Francisco, California (January 2008)
4. Brabrand, C., Møller, A., Schwartzbach, M.I.: Dual syntax for XML languages. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774. Springer, Heidelberg (2005)
5. Braun, P., Marschall, F.: Transforming object oriented models with botl. Electronic Notes in Theoretical Computer Science, 72(3) (2003)
6. Chivers, H., Paige, R.F.: Xround: Bidirectional transformations and unifications via a reversible template language. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 205–219. Springer, Heidelberg (2005)
7. Cicchetti, A., Di Ruscio, D., Eramo, R.: Towards propagation of changes by model approximations. In: Proceedings of the Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), p. 24. IEEE Computer Society, Los Alamitos (2006)
8. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal, special issue on Model-Driven Software Development 45(3), 621–645 (2006)
9. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 72–86. Springer, Heidelberg (2007)
10. Del Fabro, M.D., Jouault, F.: Model transformation and weaving in the AMMA platform. In: Proceedings of GTTSE 2005 (2006)
11. Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency handling in multi-perspective specifications. Transactions on Software Engineering 20(8), 569–578 (1994)
12. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Transactions on Programming Languages and Systems 29(3), 17 (2007)
13. Greenyer, J., Kindler, E.: Reconciling TGGs with QVT. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 16–30. Springer, Heidelberg (2007)
14. Grunske, L., Geiger, L., Lawley, M.: A graphical specification of model transformations with triple graph grammars. In: proceedings of 2nd European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA) (November 2005)
15. Hibberd, M., Lawley, M., Raymond, K.: Forensic debugging of model transformations. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 589–604. Springer, Heidelberg (2007)

16. Hu, Z., Mu, S.-C., Takeichi, M.: A programmable editor for developing structured documents based on bidirectional transformations. In: Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation (PEPM 2004), pp. 178–189 (2004)
17. Jouault, F., Kurtev, I.: Transforming models with atl. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
18. Kawanaka, S., Hosoya, H.: biXid: a bidirectional transformation language for XML. In: Proceedings of the International Conference on Functional Programming, ICFP 2006, pp. 201–214 (2006)
19. Klar, F., Königs, A., Schürr, A.: Model transformation in the large. In: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007, pp. 285–294. ACM, New York (2007)
20. Königs, A., Schürr, A.: Tool Integration with Triple Graph Grammars - A Survey. In: Heckel, R. (ed.) Proceedings of the SegraVis School on Foundations of Visual Modelling Techniques. Electronic Notes in Theoretical Computer Science, vol. 148, pp. 113–150. Elsevier Science Publ., Amsterdam (2006)
21. Königs, A.: Model transformation with triple graph grammars. In: Proceedings of the Workshop on Model Transformations in Practice, at MODELS 2005 (September 2005)
22. Küster, J.M.: Definition and validation of model transformations. Software and Systems Modeling (SoSyM) 5(3), 233–259 (2006)
23. Lambert Meertens. Designing constraint maintainers for user interaction. Unpublished manuscript (June 1998), `http://www.kestrel.edu/home/people/meertens/`
24. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electr. Notes Theor. Comput. Sci. 152, 125–142 (2006)
25. Mu, S.-C., Hu, Z., Takeichi, M.: An algebraic approach to bi-directional updating. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 2–20. Springer, Heidelberg (2004)
26. Mu, S.-C., Hu, Z., Takeichi, M.: An injective language for reversible computation. In: Kozen, D. (ed.) MPC 2004. LNCS, vol. 3125, pp. 289–313. Springer, Heidelberg (2004)
27. Oliveira, J.N.: Data transformation by calculation. In: Informal GTTSE 2007 Proceedings, pp. 139–198 (July 2007)
28. OMG. MOF2.0 query/view/transformation (QVT) adopted specification. OMG document ptc/05-11-01 (2005), `www.omg.org`
29. Sendall, S., Küster, J.M.: Taming model round-trip engineering. In: Proceedings of Workshop on Best Practices for Model-Driven Software Development, Vancouver, Canada (2004)
30. Stevens, P.: Bidirectional model transformations in qvt: Semantic issues and open questions. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 1–15. Springer, Heidelberg (2007)
31. Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovsky, T., Prange, U., Varro, D., Varro-Gyapay, S.: Model transformation by graph transformation: A comparative study. In: Proceedings of the Workshop on Model Transformations in Practice, at MODELS 2005 (September 2005)
32. Witkop, S.: MDA users' requirements for QVT transformations. OMG document 05-02-04 (2005), `www.omg.org`