Correctness of Incremental Model Synchronization with Triple Graph Grammars

Fernando Orejas [⋆1] and Elvira Pino[1]

Universitat Politècnica de Catalunya, Spain
{orejas, pino}@lsi.upc.edu

Abstract. In model-driven software development, we may have several models describing the same system or artifact, by providing different views on it. In this case, we say that these models are consistently integrated.
Triple Graph Grammars (TGGs), defined by Schürr, are a general and powerful tool to describe (bidirectional) model transformations. In this context, model synchronization is the operation that, given two consistent models and an update or modification of one of them, finds the corresponding update on the other model, so that consistency is restored. There are different approaches to describe this operation in terms of TGGs, but most of them have a computational cost that depends on the size of the given models. In general this may be very costly since these models may be quite large. To avoid this problem, Giese and Wagner have advocated for the need of *incremental* synchronization procedures, meaning that their cost should depend only on the size of the given update. In particular they proposed one such procedure. Unfortunately, the correctness of their approach is not studied and, anyhow, it could only be ensured under severe restrictions on the kind of TGGs considered.
In the work presented, we study the problem from a different point of view. First, we discuss what it means for a procedure to be incremental, defining a correctness notion that we call *incremental consistency*. Moreover, we present a general incremental synchronization procedure and we show its correctness, completeness and incrementality.

Key words: Model Transformation, Model Synchronization, Triple Graph Grammars, Incremental Model Synchronization.

1 Introduction

In model-driven development, we may have several models describing the same system or artifact, by providing different views on it. Then, we say that these models are consistently integrated. Similarly, we say that two models are consistent if they are complementary descriptions of some system. In this context, given two integrated models, model synchronization is the problem of restoring consistency when one of these models has been updated by propagating that update to the other model. The same problem is also studied in other areas like databases or programming languages [1, 14, 9].

⋆ This work has been partially supported by the CICYT project (ref. TIN2007-66523) and by the AGAUR grant to the research group ALBCOM (ref. 00516)

Triple Graph Grammars (TGGs) [11, 12] are a general and powerful tool to describe (bidirectional) model transformations. On the one hand, a TGG allows us to describe classes of consistently integrated models and, on the other hand, given some source model M_1, using the so-called derived operational rules associated to the TGG, we can find a corresponding consistent target model M_2. There are different approaches to describe model synchronization in terms of TGGs, but most of them have a computational cost that depends on the size of the given models. This may be rather inefficient since the given models may be large. To avoid this problem, Giese and Wagner [4] have advocated for the need of *incremental* synchronization procedures, meaning that their cost should depend only on the size of the given update. In particular they proposed one such procedure. Unfortunately, the correctness of this approach is not studied and, anyhow, it could only be ensured under severe restrictions on the kind of TGGs considered, since the approach only works for the case when source and target models are bijective.

In this paper we address the problem from a different point of view. First, we discuss what it means for a procedure to be incremental. Specifically, given a derivation used to create a model and an update on it, we establish what does it means incrementality with respect to a consistent submodel not affected by the update. Essentially, it means that there exists a derivation that builds the new model preserving that consistent submodel. Then, this idea is formulated as a correctness notion, that we call *incremental consistency*. This may be considered a first contribution of the paper.

Our second and main contribution is the introduction of a new general incremental synchronization procedure. In principle, the input for this procedure would be given by an integrated model $\overline{G}$, a derivation of $\overline{G}$ representing its structure, and an update on the source model of $\overline{G}$. However, since storing a derivation may be expensive in terms of the amount of storage needed, we replace the derivation by dependence relations on the elements of $\overline{G}$ that are shown to be equivalent, in an adequate sense, to the derivation. Specifically, we prove a theorem (Th.1 in Sec.4) that guarantees that the largest consistent submodel not affected by the update can be obtained from that dependencies without cost depending on the model. Then, the procedure consists of five steps. In the first one, based on the above result, we identify the part of the model that needs to be reconstructed and we mark all the elements that may need to be deleted. In the second step, if needed, we enlarge the part of the model that needs to be reconstructed. As we will discuss, this second step is only needed in some cases when the update does not allow incremental consistency with respect to the largest consistent submodel not affected by the update, but with respect to a smaller one. In the third step, following the same idea presented informally in [5], we build a model that is already consistent, by applying a variation of forward translation rules [8, 6] allowing us to reuse most relevant information from the target model. For this reason, we call these rules *forward translation rules with reuse*. However, the resulting model may not include elements from the target model that do not have a correspondence in the source model. To avoid this, in the fourth step we recover these elements by just using our dependence relations. Finally, in the fifth step we effectively delete target elements that are still marked to be deleted. We prove that the results of this procedure are always incrementally correct and complete in the sense that, if there is an incrementally correct solution, the procedure will find it.

When describing our procedure, sometimes we refer to *user interaction* to take some decisions that may be not obvious. We want to point out that, from a theoretical point of view, this is equivalent to considering that our procedure is nondeterministic. On the other hand, it is important to notice that we do not assume any restriction on the kind of grammars or graphs considered in this paper. This is not the case of most other approaches that impose reasonable restrictions to ensure efficiency. As a consequence, the implementation of our procedure may be computationally costly since, at some points some exhaustive search may be needed. However, our ideas could also be used in the context of the restrictions considered by other authors. In that case, our procedure would be as efficient (or more efficient) than these other approaches. Anyhow, it must be understood that our contribution is related to the study of when and how we can proceed incrementally in the synchronization process in the most general case, rather than restricting its application to the cases where a certain degree of efficiency is ensured.

The paper is organized as follows. In Section 2 we introduce some basic material needed in the paper and we present a running example that we use to illustrate our approach and results. In the third section we study the notion of incremental consistency and in Section 4 we present the dependency relations that are used to represent derivations. In Section 5 we present our incremental synchronization procedure. Finally, in Section 6 we discuss related work and we draw some conclusions.

2 Preliminaries

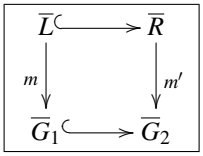
In this section, we describe some basic notions and terminology concerning, model transformation and model synchronization with Triple Graph Grammars (TGGs). Moreover, we introduce the examples that we will use in the paper.

2.1 Model Synchronization with Triple Graph Grammars

Model synchronization is the operation that, given two consistent models and an update or modification of one of them, finds the corresponding update on the other model, so that consistency is restored. Let us be more precise. First, we consider that models are some kind of typed graphs with attributes (see, e.g., [2]). This means that our models consist of nodes, edges and attributes[1], which are values associated to nodes and edges. Moreover, a type graph, which is similar to a metamodel, describes the kind of elements (nodes, edges and attributes) that are part of the given class of models. For example, in Fig. 1 we can see the type graph of the example considered in this paper. Second, we consider that integrated models are not just pairs of graphs but *triple graphs* that, in addition, provide a correspondence between elements of the given models. Formally, a triple graph $\overline{G} = (G^S \overset{s_G}{\leftarrow} G^C \overset{t_G}{\rightarrow} G^T)$ consists of a *source graph* G^S and a *target graph* G^T, which are related via a *correspondence graph* G^C and two mappings (graph morphisms) $s_G : G^C \rightarrow G^S$ and $t_G : G^C \rightarrow G^T$ specifying how source elements correspond to target elements. For example in figure 2 we can see a triple graph typed by the graph in figure 1. For simplicity, we use double arrows, $\langle G^S \leftrightarrow G^T \rangle$, as an equivalent shorter notation for triple graphs, whenever the explicit correspondence graph can be omitted.

[1] For simplicity, our example includes no attributes

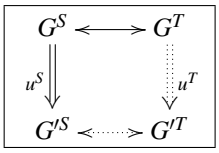
A simple but powerful way of describing a class of consistently integrated models is by using a *Triple Graph Grammar* [11, 12], consisting of a start triple graph, $\overline{SG}$[2], and a set of production rules of the form $p : \overline{L} \to \overline{R}$, where $\overline{L}$ and $\overline{R}$ are triple graphs and $\overline{L} \subseteq \overline{R}$. That is, $\mathcal{L}(\mathcal{G}) = \{\overline{G} \mid \overline{SG} \stackrel{*}{\Longrightarrow} \overline{G}\}$ is the language defined by a triple grammar $\mathcal{G}$, where $\stackrel{*}{\Longrightarrow}$ is the reflexive and transitive closure of the one step transformation relation $\Longrightarrow$ defined by the grammar as follows. $\overline{G}_1 \Longrightarrow \overline{G}_2$ if there is a production rule $p : \overline{L} \to \overline{R}$ in $\mathcal{G}$ and a matching monomorphism $m : \overline{L} \to \overline{G}_1$ such that $\overline{G}_2$ can be obtained by replacing (the image of) $\overline{L}$ in $\overline{G}_1$ by (a corresponding image of) $\overline{R}$. Formally, this means that the diagram above on the right is a pushout in the category of triple graphs. In this case, we write $\overline{G}_1 \stackrel{p,m}{\Longrightarrow} \overline{G}_2$, or just $\overline{G}_1 \Longrightarrow \overline{G}_2$ if p and m are implicit.



Hence, we say that a triple graph $\overline{G}$ is consistent if $\overline{G} \in \mathcal{L}(\mathcal{G})$. Similarly, we say that a source graph G^S (respectively, a target graph G^T) is consistent if there exists a triple graph $\langle G^S \leftrightarrow G^T \rangle \in \mathcal{L}(\mathcal{G})$.

Finally, we consider that an *update* or *modification* of a graph G, denoted $u : G \Rightarrow G'$ is a span of inclusions (or, in general, injective morphisms) $G \leftarrow G_0 \rightarrow G'$ for some graph G_0. Intuitively, the elements in G that are not in G_0 are the elements deleted by u, and the elements in G' that are not in G_0 are the elements added by u.

Now we can express formally the synchronization problem in terms of the diagram on the right [7]. Given a triple graph $\overline{G}$ and an update $u^S : G^S \Rightarrow G'^S$ on the source graph[3], the synchronization problem is to find an update $u^T : G^T \Rightarrow G'^T$ and a triple graph $\overline{G'}$ such that $\overline{G'}$ is consistent. These results, u^T and $\overline{G'}$, are called the forward propagation of u^S over $\overline{G}$. Notice that finding the triple graph $\overline{G'}$ means computing the new correspondences of that graph. Notice also that, in general, there may be no solution to the synchronization problem. In particular, this is the case if G'^S is not consistent, i.e. when there is no consistent triple graph $\langle G'^S \leftrightarrow G'^T \rangle$.



For example, below we will consider the synchronization problems when deleting the subclass edge between classes C_3 and C_2 in the triple graph in Fig. 3.

2.2 Model Transformation with Triple Graph Grammars

Model transformation is the problem of finding a consistent triple graph $\langle G^S \leftrightarrow G^T \rangle$, when given a TGG $\mathcal{G}$ and a source model G^S. This problem is very related to the problem of model synchronization. Each of these problems can be seen as a special case of the other one. In particular, the model transformation problem can be seen as a special case of model synchronization since it can be solved by computing the propagation of the update $u^S : \emptyset \Rightarrow G^S$ over the empty triple graph $\langle \emptyset \leftrightarrow \emptyset \rangle$. Similarly, model synchronization can be reduced to model transformation, since given $\langle G^S \leftrightarrow G^T \rangle$ and a source update $u^S : G^S \Rightarrow G'^S$ we can solve the synchronization problem just computing

[2] In general, without loss of generality, we will consider that $\overline{SG}$ is always the empty triple graph.

[3] Note that the synchronization problem after a target update can be seen as a special case of the problem considered in this paper, since triple graphs are symmetric structures.

the model transformation of G'^S. However, this would not be an efficient solution of the synchronization problem[4].

In the context of TGGs, the model transformation problem can be solved using the so-called operational rules (forward, backward, source, and target rules) associated to $\mathcal{G}$. The key idea is that forward rules, generated from the rules in $\mathcal{G}$, preserve the given source model but add the missing target and correspondence elements. Solving this problem is equivalent to finding a source consistent derivation [3].

$$\langle G^S \leftrightarrow \emptyset \rangle \Longrightarrow_{p_1} \langle G^S \leftrightarrow G_1^T \rangle \Longrightarrow_{p_2} \ldots \Longrightarrow_{p_n} \langle G^S \leftrightarrow G_n^T \rangle$$

where $p_1, p_2, \ldots, p_n$ are forward rules associated to $\mathcal{G}$.

Finding source consistent derivations or checking if a derivation with forward rules is source consistent is, in general, quite costly. For this reason, [6] introduces a new technique based on the use of Boolean-valued *translation attributes*. These attributes are associated to all elements in the graph (i.e. nodes, edges, and also other attributes) to denote if that element has been *created* or not by a rule. The idea is quite simple. Let us first consider a slightly different problem. Suppose that we want to know if a given triple graph is consistent, i.e. if $\overline{G} \in \mathcal{L}(\mathcal{G})$. Obviously, we may try to see if we can derive $\overline{G}$ using the rules in $\mathcal{G}$. However, we can use a different approach: we modify slightly the TGG rules so that, instead of creating new elements, we just mark the existing ones, so that to check if $\overline{G}$ is consistent, we check if we can mark all its elements with the modified rules. These marks are the translation attributes, that is, the attribute of an element states if the element has been marked or not. Then, to check if $\overline{G}$ is consistent we just have to add all the translation attributes set to `false`, and try to see if applying the modified rules we can arrive to a graph with all its translation attributes set to true.

The above idea can be generalized. Suppose that we have a grammar $\mathcal{G}$ and a (not necessarily consistent) triple graph $\overline{G}$, and we want to extend it until we arrive to a consistent graph. A straightforward approach would be to use the rules in $\mathcal{G}$ to find a graph $\overline{G'}$ that extends $\overline{G}$. But we can also modify the rules in $\mathcal{G}$, so that, if an old rule would have created an element already in $\overline{G}$, the new rule would just mark it; but if the old rule would create a new element not in $\overline{G}$, the new rule would also create it. We can say that these new rules *reuse* the elements in $\overline{G}$. A similar idea was informally introduced in [5]. For example, *forward translation rules* [6] follow this idea to solve the model transformation problem. The part that is reused is the given source graph G^S, and the extension that we are looking for consists of the target and correspondence parts of the result. That is, $\langle G^S \leftrightarrow \emptyset \rangle$ is the given triple graph that we want to complete. So, to solve the model transformation problem, we would first add `false` translation attributes to all the elements in G^S and then apply the new rules until we arrive to a triple model $\overline{G}$ whose source part is like G^S, but with all its translation attributes set to `true` .

2.3 Example

In this subsection we introduce the example that is used to illustrate our techniques. It is a simplified, and slightly modified, version of the well-known transformation between

[4] Actually it would neither be an adequate solution [13].

class diagrams and relational schemas. The type graphs of source, target and correspondence models are depicted in Fig. 1. Source models, whose type graph is depicted on the left, consist of two kinds of nodes, classes and attributes, and two kinds of edges. On the one hand, an edge between two classes represents a subclass relationship between them. On the other hand, attributes are bound to their associated classes by the second kind of edges. Similarly, the type graph of target models is depicted on the right of the figure, consisting of tables, columns and foo nodes[5], together with edges between columns and tables, and between foo nodes and columns. Finally, in the center of Fig. 1, we depict the type graph of the correspondence models, consisting of two kinds of nodes: square nodes to bind classes with their associated tables, and round nodes to bind attributes with their associated columns.

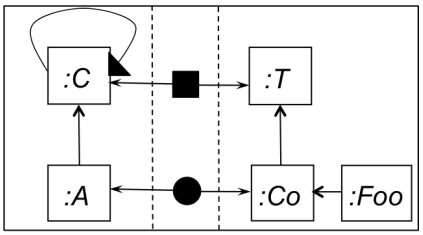


Fig. 1. Type graph

The rules of the TGG defining the transformation between class diagrams and relational schemas are depicted in Fig. 2 in short notation, i.e. left and right hand sides of a rule are depicted in a single triple graph. Elements which are created by the rule are labelled with $++$ and additionally marked by purple line colouring. Rule 1, Class2Table creates a new class and its corresponding table. and it also creates the correspondence element that relates the class and the table. Rule 2, Attribute2Column, given a class C_1 and a corresponding table T_1, creates an attribute A_1 of C_1 and a related column c_1 of T_1, together with the associated correspondence element. Rule 3, Subclass2Table, given a class C_1 and a corresponding table T_1, creates a new class C_2 that is a subclass of C_1. In this case, C_2 is related to T_1 through a new correspondence element. Finally, Rule 4, FooCreation creates a new foo node associated to an existing column. Notice that, in this rule, the source and correspondence parts of the triple rule are empty.

In the left of Fig. 3 we depict a triple graph generated by this TGG. That triple graph could have been generated by a derivation d_1 consisting of, first, applying twice the rule Class2Table, to create classes C_1, C_2 and tables T_1, T_2; then, applying the rule Subclass2Table, to create C_3, and applying three times the rule Attribute2Column to create attributes A_1, A_2, A_3 and columns c_1, c_2, c_3; finally applying the rule FooCreation to create the foo node associated to column c_3. But it could have also been created by other derivations that are *permutation equivalent* to d_1 [2], like derivation d_2, consisting of applying twice the rules Class2Table and Attribute2Column, to create classes C_1, C_2, ta-

[5] These foo nodes have no special meaning. They are just introduced for our convenience.

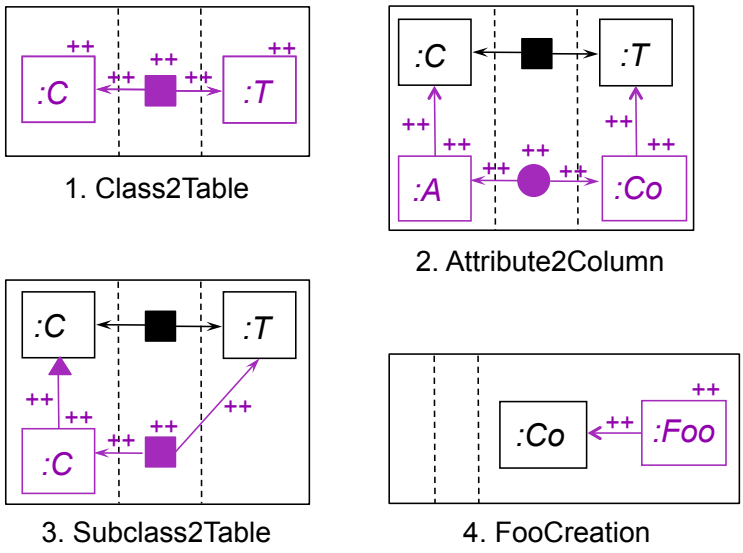


Fig. 2. Transformations Rules

bles T_1, T_2, attributes A_1, A_2, and columns c_1, c_2.; then, applying rule Subclass2Table, to create C_3, rule Attribute2Column to create attribute A_3 and column c_3, and rule FooCreation to create the foo node.

Finally, in the rest of the paper, we will use green colour[6] to depict the elements affected by an update, in contrast to the black coloured elements that are not affected by the update.

3 Incremental Model Synchronization and Incremental Consistency

In the literature on model synchronization, the term "incremental" has two possible meanings. On the one hand, in most papers, a synchronization procedure is called incremental if the propagation of a source update reuses the information included in the given target model. Actually, according to this meaning of incrementality, rather than saying if a procedure is incremental or not, we should say how much incremental it is, depending on the amount of target information reused. For instance, an extreme case would be a procedure that, given an integrated model $\overline{G}$ and a source update $u^S : G^S \rightarrow G'^S$, would compute the propagation of u by computing the model transformation of G'^S, without taking into account the information in G'^T. Obviously, this would be the most non-incremental (or the least incremental) procedure.

On the other hand, in [4], Giese and Wagner advocate that synchronization should be *incremental*, meaning that its computational cost should depend mainly on the size

[6] For readers of black and white prints, green elements appear as lighter grey.

of the modification and not on the size of the given models. This is not the case in most existing approaches. Even if they build the solution by modifying the given target model and reusing its information, their cost still depends on the size of the given models, because they have to analyze the models to ensure correctness. Our aim is to develop a procedure that is incremental in both senses.

Our approach is based on assuming that if $\overline{G}$ is the given integrated model, we know which derivation $d = \overline{SG} \Longrightarrow \ldots \Longrightarrow \overline{G_i} \Longrightarrow \ldots \Longrightarrow \overline{G}$ generated it. In this context, if we know that the given update $u^S : G^S \Rightarrow G'^S$ does not affect any element in G_i^S, i.e. $G_i^S \subseteq G'^S$ and the result of the synchronization, $\overline{G'}$, also includes $\overline{G_i}$, then we say that $\overline{G'}$ is *incrementally consistent* with respect to $\overline{G_i}$. Then, the idea underlying our procedure is to find the largest $\overline{G_i} \subseteq \overline{G}$ so that we can build over it the solution $\overline{G_i} \subseteq \overline{G'}$. Moreover, since we want our procedure to be incremental in the sense of [4], the cost of finding $\overline{G_i}$ should not depend on its size.

However, there are many derivations that can be considered equivalent, because the order in which we apply some productions is irrelevant. These transformations are called *sequentially independent* and the derivations are *permutation equivalent* (for the concrete definitions see, e.g., [2]). For instance, in the example, derivations d_1 and d_2 mentioned in subsection 2.3, are permutation equivalent. This means that it is not relevant if we first create classes C_1, C_2, C_3 and tables T_1, T_2 and then we add the attributes and columns $A_1, A_2, A_3, c_1, c_2, c_3$, or if we first create C_1, T_1, A_1, c_1, then C_2, T_2, A_2, c_2 and finally C_3, A_3, c_3, or if we create the classes, attributes, tables and columns in a different order. The only limitations are that we cannot create C_3 before C_2 and T_2, because the rule to create C_3 needs that C_2, T_2 are already there, neither we can create an attribute/column before their associated class/table, nor we can create a foo node before its corresponding column. As a consequence, when looking for the submodel $\overline{G_i}$ to build the synchronization, we must consider, not only the given derivation d that generated $\overline{G}$, but also all derivations that are permutation equivalent to d.

For example, let us suppose that, in the graph on the left of Fig. 3, we delete the subclass relation between C_3 and C_2. The result of the (expected) synchronization is depicted on the right of that figure. We may see that this result is incrementally consistent with respect to the subgraph depicted in black on the left. So, in this case, our procedure would first need to find that subgraph and, then, it would construct the result on the right.

Definition 1 (Incremental consistency). *Given a TGG $\mathcal{G}$, a derivation $d = \overline{SG} \overset{*}{\Longrightarrow} \overline{G}$ and an update $u : G^S \Rightarrow G'^S$. Let $\overline{H} \subseteq \overline{G}$ be such that no element in H^S is deleted by u and there is a derivation d_0, permutation equivalent to d with $d_0 = \overline{SG} \overset{*}{\Longrightarrow} \overline{H} \overset{*}{\Longrightarrow} \overline{G}$. We say that an integrated model $\overline{G'} = \langle G'^S \leftrightarrow G'^T \rangle \in \mathcal{L}(\mathcal{G})$ is* incrementally consistent *with respect to d, u and $\overline{H}$ if there exists a derivation $d' = \overline{SG} \overset{*}{\Longrightarrow} \overline{G'}$ sastifying that, $d' = \overline{SG} \overset{*}{\Longrightarrow} \overline{H} \overset{*}{\Longrightarrow} \overline{G'}$.*

In most cases, we may consider that the submodel $\overline{G_i} \subseteq \overline{G}$ that we look for in our procedure should always be the largest submodel of $\overline{G}$ generated by a derivation permutation equivalent to d that is not affected by the given update. This works fine in many cases, like in the example that we have just described. However, there are cases where this largest model cannot be completed to an incrementally consistent model. For

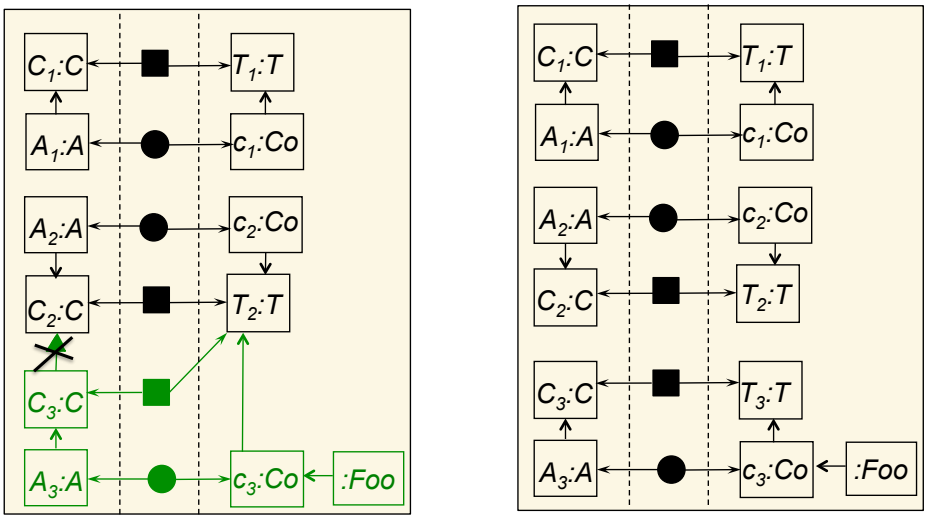


Fig. 3. Model Synchronization

instance, let us suppose that the given integrated model $\overline{G}$ is the triple graph on the right of Fig. 3 and let us suppose that the given updates consists just of the addition of a subclass relation between C_3 and C_2. In this case, the largest submodel of $\overline{G}$ not affected by that update is the whole model $\overline{G}$, since the update includes no deletion. However, there is no way to extend G^T so that the final result is consistent. In this case, the submodel of $\overline{G}$ that we can use to build an incrementally consistent result is the part of $\overline{G}$ depicted in black in the triple graph on the left of Fig. 3.

Nevertheless, the following proposition shows that, given a derivation d of an integrated model $\overline{G}$ and given a subset of elements $D \subseteq \overline{G}$ that must be deleted when applying a given update, there is a largest consistent graph $\overline{G}_{d\setminus D} \subseteq \overline{G}$ that consists of all the elements of $\overline{G}$ that can be generated by d without the use of elements from D in any derivation d' that is permutation equivalent to d. Moreover, some of these derivations d' include the derivation of $\overline{G}_{d\setminus D}$, i.e. $d' : \overline{SG} \stackrel{*}{\Longrightarrow} \overline{G}_{d\setminus D} \stackrel{*}{\Longrightarrow} \overline{G}$, then we say that d' is *maximally preserving* with respect to D.

The idea is that the result of the synchronization will be built from $\overline{G}_{d\setminus D}$. As a consequence, in most cases, D will be the set of elements deleted by the given update and, as a consequence, $\overline{G}_{d\setminus D}$ would be be the largest submodel of $\overline{G}$ generated by a derivation permutation equivalent to d that is not affected by the given update. However, as explained above, we will need to include in D some additional elements from G^S to ensure that we can extend $\overline{G}_{d\setminus D}$ to a consistent graph.

Proposition 1 (Maximal preserving derivations). *Given a derivation $d = \overline{SG} \stackrel{*}{\Longrightarrow} \overline{G}$ and a subset of elements $D \subseteq \overline{G}$, there is a consistent graph $\overline{G}_{d\setminus D} \subseteq \overline{G}$ such that, for every derivation d' permutation equivalent to d, if $d' = \overline{SG} \stackrel{*}{\Longrightarrow} \overline{G}_i \stackrel{*}{\Longrightarrow} \overline{G}$ and $\overline{G}_i$ does not include any element from D then $\overline{G}_i \subseteq \overline{G}_{d\setminus D}$. Moreover, there are derivations*

$d' = \overline{SG} \overset{*}{\Longrightarrow} \overline{G}_{d\setminus D} \overset{*}{\Longrightarrow} \overline{G}$ *which are* maximally preserving *and permutation equivalent to* d.

The idea of the proof is quite simple. The consistent submodel $\overline{G}_{d\setminus D}$ is built by including in $d_0 = \overline{SG} \overset{*}{\Longrightarrow} \overline{G}_{d\setminus D}$ all transformations in d that are sequentially independent with respect to the productions in d that need from elements in D. Then, d' is built by extending d_0 with the remaining transformations in d.

4 Derivation Dependencies

To build incrementally consistent solutions we need information about the derivation that generated the given integrated model, since we need to know what part of the model must remain unchanged after update propagation. However, saving derivations and working with them may be costly and cumbersome. In this section, we show that just saving some dependence information associated to the given derivation is enough for our purposes. The basic idea is to define some dependency relations between the elements (nodes, edges and attributes) of the given integrated model $\langle G^S \leftrightarrow G^T \rangle$ that describe if an element e_1 was needed for the creation of e_2 in a given derivation d. The first relation, called *strict dependency* and denoted $e_1 \lhd_d e_2$, holds if e_1 was matched by the left-hand side of the rule that created e_2. For instance, in Example 2, $C_2 \lhd_{d_1} C_3$ and $T_2 \lhd_{d_1} C_3$, since the application of rule Subclass2Table that created C_3 in derivation d_1 had to match its lefthand side to C_2 and T_2 (and also to their correspondence element). The second relation, called *interdependency* and denoted $e_1 \bowtie_d e_2$, holds if e_1 and e_2 are created by the same rule. For instance, in Example 2, $C_2 \bowtie_{d_1} T_2$, since they are both created by the same application of the Class2Table rule in d_1. Obviously, C_2 and T_2 are also interdependent with their correspondence node.

Definition 2 (Dependency relations). *Given* $\mathcal{G}$ *and a derivation* $d : \overline{SG} \overset{*}{\Longrightarrow} \overline{G}$*, we define the following relations on elements of* $\overline{G}$*:*

1. *Strict dependency:* $\lhd_d$ *is the smallest relation satisfying that if* d *includes the transformation step depicted below:*

$$\begin{array}{ccc} \overline{L} & \hookrightarrow & \overline{R} \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle m'} \\ \overline{G}_{i-1} & \hookrightarrow & \overline{G}_i \end{array}$$

   then for every e *in* $\overline{L}$ *and* e' *in* $\overline{R} \setminus \overline{L}$*,* $m(e) \lhd_d m'(e')$*.*
2. *Strict interdependency:* $\bowtie_d$ *is the smallest relation satisfying that if* d *includes the transformation step depicted above, then for every* e, e' *in* $\overline{R} \setminus \overline{L}$*,* $m'(e) \bowtie_d m'(e')$*.*

The key result for our synchronization procedure is the following theorem that shows that if in the given integrated model we delete the set of elements D that are dependent on a given update, the resulting triple graph is $\overline{G}_{d\setminus D}$. Moreover, it also shows that if we are interested in any submodel of $\overline{G}_{d\setminus D}$ that is also generated by the given derivation (or some permutation equivalent derivation), it is enough to remove from $\overline{G}_{d\setminus D}$ some additional elements together with all the elements that depend on them.

Theorem 1 (Dependency Relations and Incrementality). *Let $d = \overline{SG} \overset{*}{\Longrightarrow} \overline{G}$ be a derivation, D be a the subset of elements of $\overline{G}$, and $Clos_d(D)$ be the least set satisfying that:*

- $D \subseteq Clos_d(D)$,
- *If $e' \in Clos_d(D)$ and $e' \prec_d e$ or $e' \bowtie_d e$ then $e \in Clos_d(D)$,*

then:

1. $\overline{G}_{d\setminus D} = \overline{G} \setminus Clos_d(D)$.
2. *Given a subset D_0 of elements of $\overline{G}_{d\setminus D}$, there is a derivation $d_0 = \overline{SG} \overset{*}{\Longrightarrow} (\overline{G} \setminus Clos_d(D)) \setminus Clos_d(D_0) \overset{*}{\Longrightarrow} \overline{G}$ permutation equivalent to d.*
3. *Conversely, if $\overline{H}$ is a submodel of $\overline{G}_{d\setminus D}$ such that there is a derivation $d_0 = \overline{SG} \overset{*}{\Longrightarrow} \overline{H} \overset{*}{\Longrightarrow} \overline{G}$ permutation equivalent to d, then there is a subset D_0 of elements of $\overline{G}_{d\setminus D}$ such that $\overline{H} = (\overline{G} \setminus Clos_d(D)) \setminus Clos_d(D_0)$.*

The proof of this theorem is not difficult. The key issue is to show that the elements in $Clos_d(D)$ or of $Clos_d(D \cup D_0)$ are exactly the elements generated by the last transformations in a maximally preserving derivation as constructed in Prop.1.

5 A Procedure for Incremental Model Synchronization

In this section we present our procedure for incremental synchronization and we show its correctness. The input for this procedure is, not only the given integrated model and source update, but also the dependency relations. Moreover, we assume that there is a translation attribute set to `true` for every element in the model. This allows us to use our techniques needed to ensure the incrementality of solutions. Then, the output is, not only the resulting integrated model (and the resulting update), but also the updated dependence relations, so that they can be used to deal with further updates. Notice that handling explicitly the dependence relations and the translation attributes is not costly, neither in space nor in time, since all this information is boolean.

According to Theorem 1, we could consider a quite simple incremental synchronization procedure. For instance, if we know that we can build an incremental solution from $\overline{G}_{d\setminus D}$, where D is the set of elements deleted by the given source update u, we could proceed as follows. In a first step, the procedure would delete from G^T and G^C all the elements depending on the elements deleted by u and would mark as non-created all source elements added by u and all source elements depending on the elements deleted by u (except the deleted elements themselves). Then we would apply forward translation rules to the resulting model until arriving to a consistent $\overline{G'}$. The dependency relations would be updated accordingly. Unfortunately, this procedure would not work as we would like as the following example shows.

Let us consider again the deletion of the subclass relation between C_3 and C_2 in the triple graph depicted on the left of Fig. 3. The result of the first step is depicted on the left of Fig. 4, where the source elements marked as non-created are depicted in green. The reason is that all the deleted target and correspondence elements depend on the

creation of C_3 as a subclass of C_2. Finally, the result after applying forward translation rules is depicted on the right of Fig. 4. As we may see, the foo element related to column c_3 is now not present. Moreover, since c_3 has been deleted and created again, if it included some additional information (e.g. some data attributes), this information may have been lost.

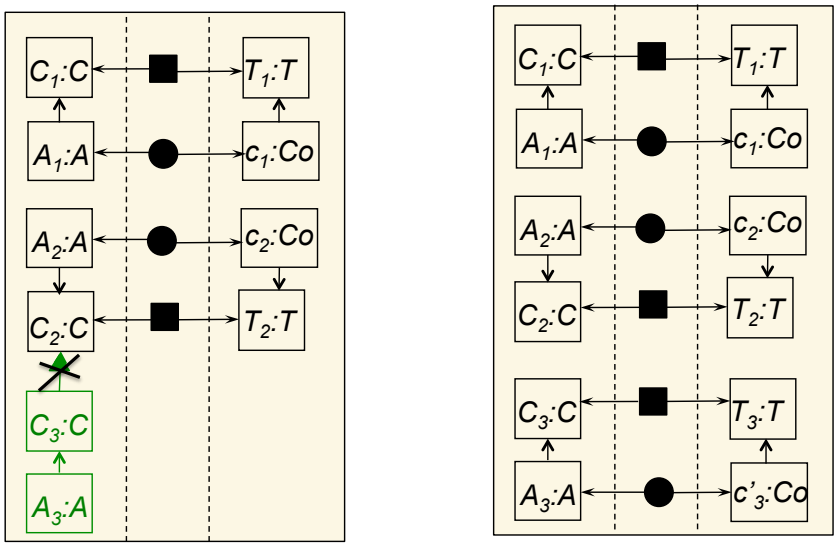


Fig. 4. Model Synchronization

To avoid these problems, we apply three ideas. The first one, quite obvious, is that the elements in the target (and correspondence) graphs should not be deleted but marked, in order not to lose information. Only at the end of the process we should delete some of these elements. The second idea is that, when building the resulting model, we should reuse the information included in the model, using a general form of forward translation rules, that use the idea presented in [5]. For this purpose, the following notion of *forward transformation rule with reuse* plays a main role, where $RemAttr(\overline{G})$ denotes the graph resulting from removing from $\overline{G}$ all its translation attributes:

Definition 3. *Given a rule $p : \overline{L} \to \overline{R}$, we say that $p' : \overline{L'} \to \overline{R'}$ is a* forward transformation rule with reuse *over p if:*

1. $RemAttr(\overline{R'}) = \overline{R}$, $\overline{L} \subseteq RemAttr(\overline{L'}) \subseteq \overline{R}$.
2. $R^S \subseteq RemAttr(L'^S)$.
3. *The translation attributes in $\overline{L'}$ are* `true` *for all the elements in $\overline{L}$, otherwise they are* `false`.
4. *All translation attributes in $\overline{R'}$ are* `true`.

The intuition of these new rules is based on the idea that the given graph $\overline{G}$ includes some elements with translation attribute `true`, which are elements considered really

in the graph, and some other elements with `false` attribute, meaning that they have not yet been created, i.e. they are not *real* elements of $\overline{G}$. So, in a rule with reuse, $\overline{L'}$ includes all elements in $\overline{L}$ with attribute `true` since, to apply the rule, all these elements must really be in $\overline{G}$. But $\overline{L'}$ may also include some elements from $\overline{R} \setminus \overline{L}$, with `false` attribute, that are reused. Then, after applying p', all the reused elements have now a `true` attribute, since they are now real elements of the graph, and all the elements in $\overline{R} \setminus \overline{L}$ which are not in $\overline{L'}$ (i.e. they have not been reused) are added to the graph with `true` attribute. Condition 2. states that all source elements in $\overline{R}$ must be in $\overline{L'}$, the reason is that these are forward rules, i.e., we assume that the rules should only add target and correspondence elements.

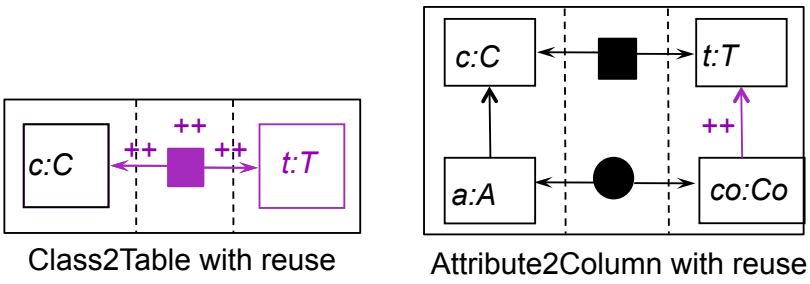


Fig. 5. Examples of rules with reuse

We may notice that, for each original rule, we may have quite a big amount of associated rules with reuse. This means that creating a priori all of them can make the model transformation and synchronization processes quite costly. Instead, we believe that, for implementation, the right approach is to work directly with the original triple graph rules, $\overline{L} \to \overline{R}$, as proposed in [5]. The idea would be that after finding a match between $\overline{L}$ and the given graph $\overline{G}$, we check how much of $\overline{R}$ we can reuse, and we proceed accordingly. However, we must warn that, in some situations, if we reuse as much as possible, some of this reuse could be inadequate. For instance, suppose that in our example the given triple graph includes also a class C_0 and a related table T_0 and suppose also that the given update not only deletes the subclass relation between C_3 and C_2, but it also includes the deletion of the class C_0. Then, with maximal reuse, instead of creating a new table T_3 and associating it to C_3, we would reuse T_0, associating it to C_3. This is probably wrong according to what the user expects, even if the result is technically correct. Anyhow, we believe that, in the worst case, it is better to produce an inadequate result that the user can easily amend, that producing some result, which is also inadequate, but where some information has been lost and can be difficult to recover. In any case, we also believe that, in general, the decision on how much to reuse should not be automatic, but it should be taken by the user.

The third idea is related to rules like FooCreation on figure 2, that includes no source elements. The model transformation process, to construct the synchronization, is driven by the source elements of the given graph. This means that, while there are source

elements with `false` translation attributes, we look for a rule that would transform some of these `false` source attributes into `true`. So, a rule like FooCreation will never be applied in this process. The problem is to know when to apply this kind of rules. The solution is given by our dependence relations. If there is a target element e with `false` translation attribute (i.e. the element was in the original model, but the previous process has not created it); if all elements e' such that $e' \triangleleft e$ have `true` attribute (i.e. the elements that were used to create e have already been created); and if all elements e'', such that $e \bowtie e''$, have `false` attribute (i.e. the other elements created together with e have not been created either); then we can turn the translation attributes of e and all the elements e'' to `true`, because this is like applying the same rule that created e. We call this operation the *recreation of e and all its interdependent elements*.

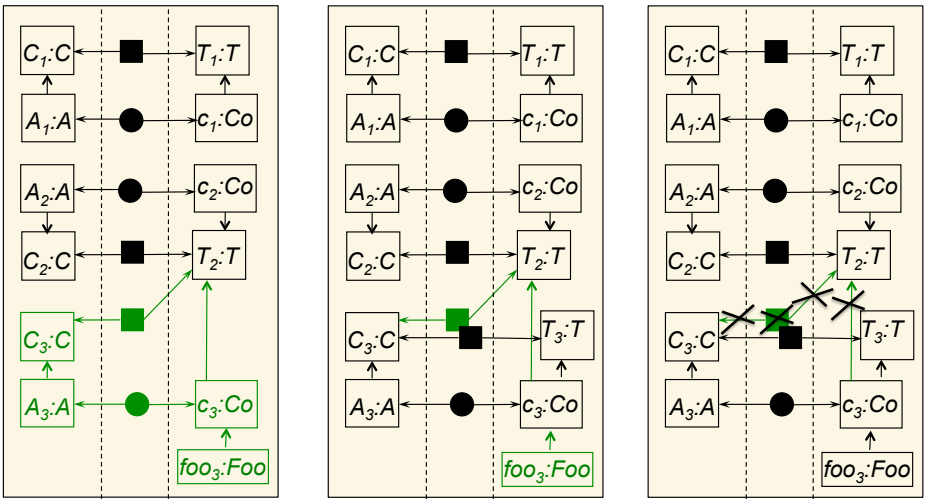


Fig. 6. Synchronization: Some intermediate steps and final model

Following the above ideas, our procedure for incremental synchronization has five steps. As said above, we assume given the original integrated model, $\overline{G} = \langle G^S \leftrightarrow G^T \rangle$, including translation attributes set to `true` for all its elements, an update $u : G^S \Rightarrow G'^S$, and the dependency relations associated to the derivation that created $\langle G^S \leftrightarrow G^T \rangle$.

1. **Updating and Marking** All elements deleted by u are deleted from G^S. All elements added by u are added to G^S with `false` translation attribute. All elements in $\overline{G}$ that are dependent on the elements deleted are marked as false. Finally, any correspondence element whose associated source element has been deleted, is deleted. As a consequence of Theorem 1, all the elements with true translation attribute form the submodel $\overline{G}_{d \setminus D}$, where D is the set of elements deleted by u.
2. **Selection of a submodel of** $\overline{G}_{d \setminus D}$. If needed, the translation attributes of some other elements of $\overline{G}_{d \setminus D}$ and the elements depending on them are set to false. This step is needed in the case where we cannot build an incrementally consistent derivation

out of $\overline{G}_{d\setminus D}$. For instance, in the case where $\overline{G}$ is the triple graph on the right of Fig. 3 and u consists of the addition of a subclass relation between C_3 and C_2, we would need to set to false the attributes of C_3 and all the elements depending on it. The decision of which translation attributes have to be set to false may be either taken by the user or by some search procedure based on some heuristics or just on backtracking.

3. **Forward Model Transformation** While there are elements in G^S with `false` translation attribute, select a rule that can match at least one of these elements. Select the amount of the rule to be reused and apply it. [7]
4. **Recreation of Target Elements** For each element in G^T with `false` translation attribute, try to recreate it together with all its interdependent elements.
5. **Deletion** Any target or correspondence element with `false` translation attribute must be deleted.

For instance, Fig. 6 depicts the results of some steps of our procedure if $\overline{G}$ is the triple graph on the left of Fig. 3 and u consists of the deletion of the subclass relation between C_3 and C_2. In particular, on the left of Fig. 6 we can see the resulting triple graph after step 1, where the green color means that those elements have a false translation attribute. In this case, step 2 causes no effect, since it is not necessary to mark to `false` any other element. In the middle figure we can see the resulting graph after step 3, where some existing elements have been reused by applying the rules Class2Table and AttributeColumn with reuse in Fig. 5. The process is almost finished, except that the Foo element has not yet been created (this is done in step 4) and we still have an edge between c_3 and T_2, and a correspondence between C_3 and T_2, both with `false` translation attribute. That edge will be deleted in step 5. Finally, on the right of Fig. 6, we have the final synchronized model after applying steps 4 and 5.

As a consequence of how our procedure works and our previous results, our procedure is incremental and incrementally correct, in the sense that the solution obtained is incrementally consistent with respect to the submodel chosen in step 2. Moreover, the procedure is complete, in the sense that if there is an incrementally consistent solution with respect to some submodel, the procedure will find it. Finally, the cost of the procedure is independent of the size of the submodel chosen in step 2.

Theorem 2. *The procedure is incremental, incrementally correct and complete.*

6 Related Work and Conclusion

As said in the introduction, model synchronization[8] is a problem studied in different areas in Computer Science. In particular, in databases (e.g., [1]), programming languages (e.g., [9]) and in model-driven software development (MDD). In the former two areas the kind of models considered are very specific, however in the latter area the kind of

[7] In general, some choices may not lead to a result where all elements in the source part have `true` translation attribute. In that case, this step may need backtracking.

[8] In some cases model synchronization is called incremental model transformation, or just model transformation. Obviously, this is quite confusing.

models considered may be very different. For this reason, in MDD we need general approaches, as TGGs [11, 12], that can be used for dealing with most kinds of models.

There are several approaches based on TGGs that propose a solution to the model synchronization problem (that we know [4, 7, 5, 10] and some variations on them) but all of them are, in our opinion, not completely satisfactory. In particular, even if the construction of the solution does not start from scratch but from the given integrated model $\overline{G}$, the approach in [7] has to analyze the complete graph $\overline{G}$ to know what parts must be modified, so its cost depends on the size of the given model. In addition, in [7] not all elements of the original graph that could be reused are indeed reused. In particular, in our example, column c_3 would have been deleted and created again. This means that, if that column would have included some additional information, this information would have been lost. Moreover, the foo element associated to c_3 would not be present in the final result. On the other hand, the only restriction considered in [7] is that the given TGG should be deterministic, to ensure that their procedure is deterministic.

The approach in [4] does not need to analyze the complete graph $\overline{G}$ to check which parts must be modified, so its cost only depends on the size of the modification. However, their approach only works for the case when source and target models are bijective, which excludes the case where source models are views of target models (or vice versa). Moreover, rules like FooCreation, with empty source graph, are forbidden. In addition, this approach shares with [7] the information loss problem. Finally, that approach has not been fully formalized.

The approach in [5] proposes a technique to avoid the loss of information in [4] that is essentially similar to our forward rules with reuse. Unfortunately, even if it is based on [4], it needs to analyze the complete graph $\overline{G}$ to check which parts must be modified, so its cost depends on the size of $\overline{G}$. Moreover, the approach imposes the same restrictions as [4] and lacks formality.

Finally, in [10], like us, the authors use precedence relations to avoid having to analyze the complete graph $\overline{G}$ to find which parts must be modified. However, their relation is coarser than ours. The reason is that our relations are directly based on a given derivation while in [10], their relation is based on the dependences established by the rules of the TGG. In particular, this means that two given elements of a model may be independent, but their relation may say that one depends on the other. This has some important consequences. In particular, their synchronization procedure only works if the given triple graph is *forward precedence preserving* and if, when adding new elements, the resulting precedence graph includes no cycles. In addition, to ensure correctness, the approach also requires that the given TGG is *source-local complete*. On the other hand, the procedure needs to use a data structure that encodes how the given graph $\overline{G}$ has been derived with the given TGG. No details are given about this structure, but we suppose that it is more complex than our dependency relations. Finally, this approach also shares with [7] the information loss problem.

To conclude, in this paper we have presented a new approach for incremental model synchronization based on TGGs that has been shown to be incremental, correct and complete. Moreover, our approach is general, in the sense that we do not restrict the class of TGGs considered. As pointed out in the introduction, we do not assume any restriction on the kind of grammars or graphs, as other approaches does. On the con-

trary, we have focussed on the study of when and how we can proceed incrementally in the synchronization process in the most general case, rather than on finding out specific conditions and limitations on graphs and grammars that could make some techniques more efficient. As a consequence, it is difficult to provide an accurate evaluation of its performance: for some TGGs our procedure may exhibit an exponential (on the size of the updated part) behavior. But for the kind of more restricted TGGs, as the ones considered in other approaches, the behavior could be close to linear. Anyhow, what obviously remains to be done is to implement the approach and evaluate it in practice.

Acknowledgements The authors would like to thank the reviewers of this paper, whose comments have contributed to improve it.

References

1. Dayal, U., Bernstein, P.A.: On the Correct Translation of Update Operations on Relational Views. ACM Trans. Database Syst. 7(3), 381–416 (1982)
2. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs of Theoretical Comp. Sc., Springer (2006)
3. Ehrig, H., Ehrig, K., Hermann, F.: From model transformation to model integration based on the algebraic approach to triple graph grammars. ECEASST 10 (2008)
4. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and System Modeling 8(1), 21–43 (2009)
5. Greenyer, J., Pook, S., Rieke, J.: Preventing information loss in incremental model synchronization by reusing elements. In: ECMFA 2011. Lecture Notes in Computer Science, vol. 6698, pp. 144–159. Springer (2011)
6. Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Formal analysis of model transformations based on triple graph grammars. Software and System Modeling To appear (2012)
7. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on Triple Graph Grammars. In: MODELS 2011. Lecture Notes in Computer Science, vol. 6981, pp. 668–682. Springer (2011)
8. Hermann, F., Ehrig, H., Orejas, F., Golas, U.: Formal analysis of functional behaviour for model transformations based on triple graph grammars. In: ICGT 2010. Lecture Notes in Computer Science, vol. 6372, pp. 155–170. Springer (2010)
9. Hofmann, M., Pierce, B.C., Wagner, D.: Symmetric lenses. In: POPL 2011. pp. 371–384. ACM (2011)
10. Lauder, M., Anjorin, A., Varró, G., Schürr, A.: Efficient model synchronization with precedence triple graph grammars. In: ICGT 2012. Lecture Notes in Computer Science, vol. 7562, pp. 401–415. Springer (2012)
11. Schürr, A.: Specification of graph translators with triple graph grammars. In: WG '94. Lecture Notes in Computer Science, vol. 903, pp. 151–163. Springer (1994)
12. Schürr, A., Klar, F.: 15 years of triple graph grammars. In: ICGT 2008. pp. 411–425 (2008)
13. Stevens, P.: Towards an Algebraic Theory of Bidirectional Transformations. In: ICGT'08. Lecture Notes in Computer Science, vol. 5214, pp. 1–17. Springer (2008)
14. Terwilliger, J.F., Cleve, A., Curino, C.: How Clean Is Your Sandbox? - Towards a Unified Theoretical Framework for Incremental Bidirectional Transformations. In: ICMT 2012. Lecture Notes in Computer Science, vol. 7307, pp. 1–23. Springer (2012)