View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Electronic Communications of the EASST (European Association of Software Science and...

Electronic Communications of the EASST
Volume 57 (2013)

Proceedings of the Second International Workshop on Bidirectional Transformations (BX 2013)

On Propagation-Based Concurrent Model Synchronization

Fernando Orejas Artur Boronat Hartmut Ehrig Frank Hermann Hanna Schölzel

19 pages

Guest Editors: Perdita Stevens, James F. Terwilliger
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

On Propagation-Based Concurrent Model Synchronization

Fernando Orejas[1] * **Artur Boronat**[1 2] † **Hartmut Ehrig**[3] **Frank Hermann**[4] **Hanna Schölzel**[3]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain.

[2] The University of Leicester, UK

[3] TU Berlin, Germany

[4] Interdisciplinary Center for Security, Reliability and Trust,
Université du Luxembourg, Luxembourg

Abstract: The aim of concurrent model synchronization is to merge pairs of updates on interrelated models. For instance, this situation may occur in the context of model driven software development when the work is distributed between different teams. A first problem is that, if the updates are in conflict, this conflict is never explicit. The reason is that the updates do not interfere directly since they are assumed to modify different models. For this reason, detecting and solving conflicts becomes already more difficult than in the more standard case of synchronizing concurrent updates over a given model. Existing general approaches define the solution to this problem in terms of the solution to the simpler problem of update propagation in bidirectional model transformation. We call these approaches *propagation based*.

In this paper, we first state some properties that, in our opinion, must be satisfied by a concurrent synchronization procedure to be considered correct. Then, we show how to check whether the given updates are conflict-free and, in this case, we present a correct synchronization procedure based on this check. Finally, we consider the case where the given updates are in conflict and we show how we can build solutions that satisfy some of the correctness properties but, in general, not all of them. Specifically, we present counter-examples that show how some of these properties may fail.

Keywords: Model synchronization, bidirectional model transformation, model-driven development

1 Introduction

The aim of concurrent model synchronization is to merge pairs of updates on interrelated models. For instance, this situation may occur in the context of model-driven software development when different teams are working on different (related) models of the same software artifact. This

* 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).
† Supported by a Study Leave from University of Leicester.

problem is a generalization of (standard) model synchronization, where updates on one model must be propagated to its related model. In particular, in standard model synchronization we look for operations that implement this propagation. However, concurrent model synchronization is more difficult due to the possible existence of conflicts between the given pair of updates. An additional complication comes from the fact that these conflicts are not explicit, since the updates are applied to different models, but they are only apparent when trying to propagate the effects of the given updates.

The only approaches [XSHT09, XSHT13, HEEO12] that we know that provide a general solution to this problem define concurrent synchronization procedures in terms of the propagation operations defined for solving the standard synchronization problem. In [XSHT13], the authors present a procedure for synchronizing conflict-free concurrent updates that roughly works as follows. Given two interrelated models M and N and two updates u and v on M and N, respectively, in the first step the procedure propagates u to N in order to check if there are conflicts with v. This is easy to see, since both v and the propagation of u, $prop(u)$, are updates on the same model. Then, if there are no conflicts the procedure merges v and $prop(u)$ and propagates back the result to M. If the procedure finds a conflict, or some other problem, then the procedure reports an error. The approach presented in [HEEO12] is similar, but allowing for the possibility of handling conflicts using results from [EET11]. In this paper, we say that this kind of approaches are *propagation-based*, because their solution is defined in terms of the propagation operations defined for standard synchronization. As we have seen, their main advantage is that, since we can put together the given updates and their propagation, we can easily detect conflicts by making them explicit and, so, we can decide how to handle them. Another advantage is that these procedures are quite generic: we define them independently of how standard synchronization procedures are defined. However, as we will see in this paper, this approach has also some limitations. For instance, propagation-based procedures are not hippocratic in general.

In this paper, we study at a general level the correctness and the limitations of propagation-based concurrent synchronization. For this purpose, we provide an abstract and formal view for propagation based synchronization that covers different existing approaches. Since some of the well established properties are too restrictive for several applied cases, we present new properties that overcome these restrictions in an elegant way. Moreover, we present formal properties concerning additional relevant aspects such as maximal preservation of given updates and soundness (avoidance of side effects). Then, we show how we can check whether the given updates are conflict-free and, in this case, we also show how we can construct a correct synchronization (Theorem 1). Moreover, when the given updates are in conflict, we present a propagation-based procedure for concurrent synchronization and show its correctness concerning most of the properties (Theorem 3). Finally, we show that existing solutions satisfy some of the correctness properties but, in general, not all of them. Specifically, we present counter-examples that show how some of these properties may fail.

The paper is organized as follows. In Section 2, we present an example of model transformation that will be used to show specific counter-examples in the rest of the paper. In Section 3, we introduce the basic theoretical framework used in the paper, first our notion of model update and, then, our assumptions about update propagation operations. In Section 4, we introduce the problem of concurrent synchronization and the properties that we claim that propagation-based procedures should satisfy. In Section 5, we present different propagation-based solutions to the

concurrent synchronization problem, first for the conflict-free case and, then, for the general case. Finally, in Section 6, we present some related work and present some conclusions.

2 Running Example

To help in providing some explanations and counter-examples we use a very simple example. Source and target models describe different views of (part of) the information system of a company. They both describe information about their employees. Following the metamodels, depicted in Fig. 1, the source metamodel, on the left, describes the employees of a certain branch of the given company, for instance the employees of the Barcelona branch. It consists just of a single class, Employee, with attributes *name* (the name of the employee), *dept* (the name of the department where the employee works), *base* (the base salary of the employee), and *bonus*. The target metamodel, depicted on the right, describes the employees of the whole company. In particular, the metamodel includes three classes: Employee, Dept and Branch. Branch and Dept include just a *name* attribute, but Employee includes three attributes: *name*, *salary* (the total salary received by the employee) and *address* (the address of the employee). In addition, each employee must be associated to the branch and to the department where he currently works.

A source and a target model are consistent if the following conditions hold:

- Every employee in the source model is also present in the target model and is associated to the Barcelona branch. Conversely, every employee associated to the Barcelona branch in the target model must also be present in the source model.
- The salary of each employee associated to the Barcelona branch in the target model is the addition of the base and bonus of that employee in the source model.
- The department of an employee in the source model is a string D if and only if that employee is associated to the department of name D in the target model.

We may notice that source and target models share some information, but also include some information of their own. For instance, source models include more information about the salaries of the employees. Conversely, target models include the address of employees, and they also include information about the employees of other branches different than Barcelona.

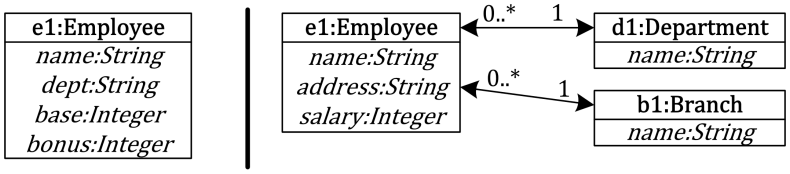


Figure 1: Source and target metamodels

Moreover, we assume that updates on the source or target models are propagated as follows:

- If we add an employee to the source model, this modification is propagated to the target model by including that employee with the address ”XXX” and with a salary that is

the sum of its base and bonus. Moreover, if the department of the new employee is not present in the target model, then the new department is also added. Finally, the employee is associated with its department and the Barcelona branch.

- Conversely, if an employee is added to the target model and it is associated to the Barcelona branch, the modification is propagated to the source model by adding that employee together with the corresponding department. Moreover, we set the base and bonus attributes to 3/4 and 1/4 of the salary, respectively.

- Finally, if an employee is deleted from either the source or the target model, its propagation consists of the deletion of that employee from the other model (if present), without any additional side-effects, like the deletion of its associated branch or department. How that employee is identified depends on specific details of the given models. For instance, if the models would be represented by a triple graph [SK08] then there would be a correspondence element relating each employee in the source model to the same employee in the target model. Otherwise, probably a key would identify each employee on the two models.

In this scenario, source and target updates occurring in parallel can cause several types of conflicts, which have to be resolved by a concurrent synchronization operation. Consider, e.g., the situation depicted in Fig. 2. The source update u^S deletes employee e_1 and the target update u^T modifies the address of this person. Propagating u^S to the current state of the target domain would remove that person. On the other hand, propagating u^T to the source domain would undo the deletion and create a new person with a base salary of 6000 and a bonus salary of 2000. There are several possibilities on how the synchronization works in the concurrent case to handle such conflicts and we will study different properties of concurrent synchronization in this paper.

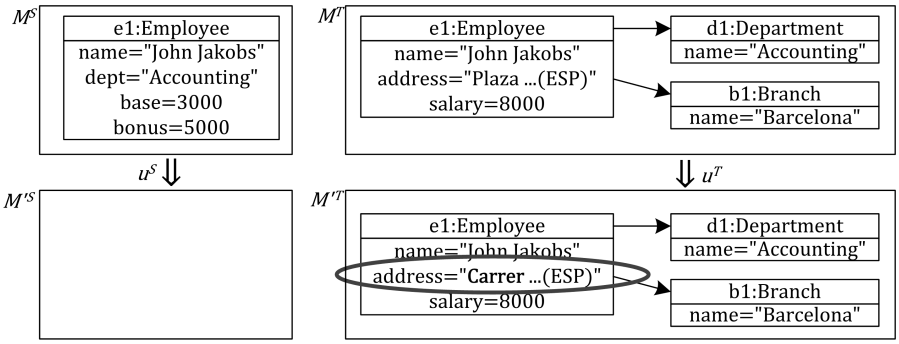


Figure 2: A parallel update

3 The Basic Framework

This paper is not about some specific kind of models. Instead, we aim to study some properties and constructions about a wide class of models and model transformations. Obviously, there

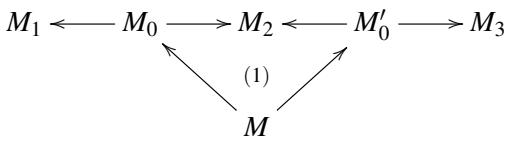
could be some frameworks where our results do not completely apply. In this sense, first, we present our assumptions about the given model framework and, then, we present a generic notion of model update or modification, and study some properties of this notion. Thereafter, in the second subsection, we present our framework of integrated models and the description of the operations of update propagation, together with the properties that we may expect that they satisfy.

3.1 Models and updates

We assume that, technically, our model classes form M-adhesive categories [LS05, EEPT06, EGH10], which roughly means that our models are some kind of structured sets where union, intersection and set difference can be defined in terms of pushouts, pullbacks and pushout complements. For example, not only sets, but most classes of graphical structures form M-adhesive categories, e.g., plain graphs, typed attributed graphs, hyper graphs, petri nets, and high-level petri nets.

Our approach is δ-*based* [DXC10], which means that, when considering the update of a model, we do not take into account only the models before and after the update. Instead, we deal with updates representing explicitly which elements have been added to or deleted from the given model (for a discussion on the advantages of δ-based approaches in comparison to state based approaches, see [DXC10]). More precisely, an *update* or *modification* [EET11] of a model M, denoted $M \Rightarrow M'$ is a span of inclusions (or, in general, injective morphisms) $M \leftarrow M_0 \rightarrow M'$. Intuitively, the elements in M that are not in M_0 are the elements deleted by the modification, and the elements in M' that are not in M_0 are the elements added by the modification. So, M_0 consists of all the elements of M that remain invariant after the modification. Updates $u_1 : M_1 \Rightarrow M_2$ and $u_2 : M_1 \Rightarrow M_2$ may be equivalent, denoted $u_1 \sim u_2$, in the sense that they yield the same result, but that they are technically different, because they are defined as different spans. For instance, the identity update $id : M \leftarrow M \rightarrow M$ is equivalent to any update $u : M \leftarrow M_0 \rightarrow M$, but $id \neq u$ if $M \neq M_0$. Intuitively, id neither deletes nor adds any element to M, while u would delete all the elements in M which are not in M_0 and, then, it would add them again.

Updates are closed under composition, i.e., given updates $u_1 : M_1 \leftarrow M_0 \rightarrow M_2$ and $u_2 : M_2 \leftarrow M'_0 \rightarrow M_3$ there exists the composition $u_2 \circ u_1 : M_1 \leftarrow M \rightarrow M_3$ defined by the diagram below:



where (1) is a pullback (intuitively, the intersection of M_0 and M'_0), i.e. M includes all the elements of M_1 that are neither deleted by u_1 nor by u_2.

Sometimes, as we will see in the sections below, we are interested, not in the composition of updates, but in their decomposition. Intuitively, an update $u_3 : M_1 \Rightarrow M_3$ can be decomposed as $u_2 \circ u_1$, denoted $u_3 \stackrel{d}{=} u_2 \circ u_1$, with $u_1 : M_1 \Rightarrow M_2$ and $u_2 : M_2 \Rightarrow M_3$ if, on the one hand, the composition $u_2 \circ u_1$ coincides with u_3 and, on the other hand, the changes included in u_1 are not undone by u_2. In particular, this requires that u_2 neither deletes any element that is created by u_1 nor it adds back any element deleted by u_1. In other words, once an element is created, it

will survive the full update, and if an element is deleted it will remain deleted for the full update. Formally, $u_3 \stackrel{d}{=} u_2 \circ u_1$ if, in the above diagram, (1) is a pullback and a pushout. Obviously, if $u_3 \stackrel{d}{=} u_2 \circ u_1$ then $u_3 = u_2 \circ u_1$, but the converse is not necessarily true. For instance, u_1 may be like u_3 but adding some extra elements and u_2 may just delete these additional elements. In that case, we would have that $u_3 = u_2 \circ u_1$ but not $u_3 \stackrel{d}{=} u_2 \circ u_1$. Moreover, we say that u_1 *decomposes* u_3 or that u_1 is a *submodification* of u_3, denoted $u_1 \trianglelefteq u_3$, if there exists u_2 such that $u_3 \stackrel{d}{=} u_2 \circ u_1$.

We may notice that, according to the previous definitions, the only submodification of the identity update $id : M \leftarrow M \rightarrow M$ is the identity update itself. The reason is that, if there would be another submodification u_1 of id, then either u_1 would delete some element from M or it would add some new element to M. But, then, any u_2 such that $u_2 \circ u_1 = id$ would need to add or delete the corresponding elements. Formally, if in the diagram above, $M_1 = M_3 = M$, the only way that (1) is a pushout and a pullback is that $M_0 = M_2 = M'_0 = M$.

It is easy to define the inverse of an update $u : M \leftarrow M_0 \rightarrow M'$. It is enough to reverse the span, i.e. $u^{-1} = M' \leftarrow M_0 \rightarrow M$. However, we may notice that $u^{-1} \circ u \sim id$ but, in general, $u^{-1} \circ u \neq id$.

When several users want to apply different updates on the same model we need to have a merge operation that combines these modifications into a single one. This problem has been studied by several authors, but here we essentially follow the work in [EET11]. A main problem is that there may be conflicts between the given modifications. For instance, an update u_1 may specify the deletion of a certain element e_1 and u_2 may specify the addition of an element e_2 that needs the existence of e_1. This is called a delete/insert conflict in [EET11]. For example, u_1 may specify the deletion of a node in a graph model and u_2 may specify the addition of an edge e_2 that is connected to e_1. Hence, a first problem is how to detect if there are conflicts between the given modifications.

Two modifications are conflict-free, if none of them deletes an element that is needed by the other one. This condition is formalised in Definition 1. Intuitively, by defining M as the pullback of (1), we are defining M as the intersection of M'_0 and M''_0, i.e. u_3 deletes from M_0 all the elements that are deleted either by u_1 or u_2. Then, the existence of M'_1 and M'_2[1] means that no element added by u_1 or by u_2 needs the existence of an element deleted by u_2 or by u_1, respectively. Finally, if (4) is a pushout, the elements added to M by $u_1 \otimes u_2$ are the union of the elements added by u_1 and by u_2. In the case of conflict-free modifications, we can define the result of this merging as the modification $u_3 = u_1 \otimes u_2 : M_0 \leftarrow M \rightarrow M_3$, where diagram (4) is a pushout. Notice that, by construction, u_1 and u_2 are submodifications of $u_1 \otimes u_2$, as we would expect.

Definition 1 (Conflict-free updates) Let $u_1 : M_0 \leftarrow M'_0 \rightarrow M_1$ and $u_2 : M_0 \leftarrow M''_0 \rightarrow M_2$ be two modifications. Let M be constructed via the pullback (1) in Figure 3. Then, u_1 and u_2 are *conflict-free*, if there are objects M'_1 and M'_2 yielding pushouts (2), (3) and (4).

In case that u_1 and u_2 are in conflict, solving the conflicts would typically mean finding conflict-free maximal submodifications $u'_1 \trianglelefteq u_1$ and $u'_2 \trianglelefteq u_2$ and merging them, as described above, where maximality means that if $u'_1 \trianglelefteq u''_1 \trianglelefteq u_1$ and $u'_2 \trianglelefteq u''_2 \trianglelefteq u_2$ and u''_1 and u''_2 are conflict-

[1] In adhesive categories, pushout complements along monomorphisms as they appear in graph modifications are unique. This means that, if they exist, M'_1 and M'_2 are unique.

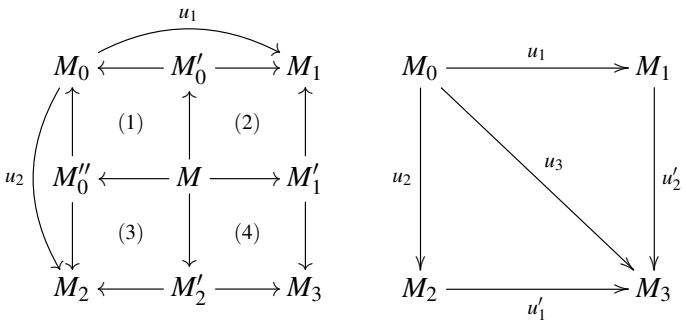


Figure 3: Condition for conflict-free modifications u_1 and u_2

free, then $u'_1 = u''_1$ and $u'_2 = u''_2$. In this paper, we are not concerned with specific procedures or strategies for solving conflicts. For example, in [EET11] a procedure is described to find solutions where delete/insert conflicts are solved giving priority to insertions.

3.2 Integrated Models and Update Propagation

We assume that two classes (categories) of models are given, $\mathcal{M}_S$ and $\mathcal{M}_T$, called the classes of source and target models, respectively, even if in a bidirectional framework there may be no specific notion of source or target. And we consider that an *integrated model* M is just a pair $M = \langle M^S, M^T \rangle$ consisting of a source and a target model. In previous work (e.g. [HEO+11, HEEO12]) we considered that an integrated model is a *correspondence*, $r : M^S \leftrightarrow M^T$, that relates the elements of both models. For many purposes, we think that working in terms of correspondences is more adequate. However, in this paper, we do not make any use of knowing the relation between the elements of both models. Therefore, for the sake of simplicity we will assume that the class of integrated models $\mathcal{IM}$ is the cartesian product $\mathcal{M}_S \times \mathcal{M}_T$.

We also assume that a subclass of consistent integrated models, $\mathcal{C} \subseteq \mathcal{IM}$ is also given. The notion of consistently integrated models induces a notion of consistence of models. In particular, a source model M_S (resp. a target model M_T) is consistent if there is a consistent integrated model $\langle M^S, M^T \rangle$ that includes that model for some target model M_T (resp. for some source model M_S).

In addition, we assume that our framework is equipped with propagation operations that solve the *basic synchronization problem*. This means that, given an integrated model M and an update on one domain, either M^S or M^T, these operations propagate the given changes to the other domain. More precisely, we assume that our framework includes suitable total forward and backward functions fPpg and bPpg. In the case of fPpg, the input is an integrated model $M \in \mathcal{IM}$ together with a source model update $u^S : M^S \Rightarrow M'^S$, and the output is a target update $u^T : M^T \Rightarrow M'^T$. The operation bPpg behaves symmetrically to fPpg. It takes as input M and a target modification $u^T : M^T \Rightarrow M'^T$ and it returns a source update $u^S : M^S \Rightarrow M'^S$. In principle, we do not require that the given integrated model, nor the result of the propagation have to be consistent, i.e. we assume that our propagation operations will provide an output for any possible update. For instance, the deletion of a given element on the source model may be propagated to the deletion of some elements of the target model, independently of the consistency of the

original and the resulting integrated models.

There are several properties that have been proposed by different authors (e.g. [Ste10, DXC+11b] to consider that propagation operations are *correct*. Below we can find the main ones. However, due to lack of space, we just state these properties for forward propagation, since the corresponding properties for backward propagation are similar. The most basic ones are the ones that we call *Consistency*, *Identity* and *Hippocraticness*. Consistency states that if the given update yields a consistent model then the resulting integrated model should be consistent.

1. **Consistency:** If $\mathsf{fPpg}(M_1, u^S : M_1^S \Rightarrow M_2^S) = (u^T : M_1^T \Rightarrow M_2^T)$ and M_2^S is consistent, then $\langle M_2^S, M_2^T \rangle$ is also consistent.

Identity states that if the given update is the identity and the given correspondence is consistent, then the propagation operations change nothing.

2. **Identity:** If $\langle M^S, M^T \rangle$ is consistent then $\mathsf{fPpg}(M, Id^S) = Id^T$

Finally, Hippocraticness states that if after the given update the resulting models are already consistent then propagation should also do nothing. Notice that identity is a special case of Hippocraticness.

3. **Hippocraticness:** Given $M_1 = \langle M_1^S, M_1^T \rangle$ and $u^S : M_1^S \Rightarrow M_2^S$, if $\langle M_2^S, M_1^T \rangle$ is consistent then $\mathsf{fPpg}(M_1, u^S) = Id^T$

Another property that has been proposed by several authors is the so called put-put law, stating the compatibility of propagation with respect to update composition:

4. **Compatibility of propagation and update composition:** If $\mathsf{fPpg}(M_1, u_1^S : M_1^S \Rightarrow M_2^S) = (u_1^T : M_1^T \Rightarrow M_2^T)$ and $\mathsf{fPpg}(M_2, u_2^S) = u_2^T$ then $\mathsf{fPpg}(M_1, u_2^S \circ u_1^S) = u_2^T \circ u_1^T$

However, the put-put law is too strong in general. For instance, in the example described in Section 2, if we delete an employee from the source model and, then, we add it again, after propagating both updates, the employee in the target model would have the address "XXX". However, the composition of the two updates is, obviously, equivalent to the identity, whose propagation is also equivalent to the identity. Hence if the address of that employee in the target model before the update was not "XXX", the put-put law would not be satisfied.

Instead of the put-put law, we propose the following alternative law that states the compatibility of propagation with respect to update decomposition:

5. **Compatibility of propagation with update decomposition:** If $\mathsf{fPpg}(M_1, u_1^S : M_1^S \Rightarrow M_2^S) = (u_1^T : M_1^T \Rightarrow M_2^T)$ and $\mathsf{fPpg}(M_2, u_2^S) = u_2^T$ and $u_3^S \stackrel{d}{=} u_2^S \circ u_1^S$ then $\mathsf{fPpg}(M, u_3^S) = u_3^T$, with $u_3^T \stackrel{d}{=} u_2^T \circ u_1^T$

Strong invertibility is a law that is also too strong in many contexts (see, e.g. [FKPT08, Ste10, DXC+11b, Ste12]. Instead, we may ask for its weaker version of *invertibility*. Strong invertibility states that if u' is the update propagation of u then u must be the update propagation (in the reverse direction) of u':

6. **Strong invertibility:** If $\mathsf{fPpg}(M,u^S) = u^T$ then $\mathsf{bPpg}(M,u^T) = u^S$

A weaker version is invertibility that roughly says that if u_2 is the update propagation of u_1 and u_3 is the update propagation (in the reverse direction) of u_2 then u_2 must be the update propagation of $u3$:

7. **Invertibility:** If $\mathsf{fPpg}(M,u_1^S) = u_1^T$ and $\mathsf{bPpg}(M,u_1^T) = u_2^S$ then $\mathsf{fPpg}(M,u_2^S) = u_1^T$

In [Ste12] the relations between some of these and other properties are studied in detail.

4 Propagation-based Concurrent Synchronization

The aim of concurrent synchronization is to merge pairs of updates (in what follows, *parallel updates*) on interrelated models. A first problem is that, if the updates are in conflict, this conflict is not explicit. The reason is that, here, on the contrary to what we have seen in Sect. 3.1, the updates do not interfere directly since they are assumed to modify different models. The conflicts may arise when we try to build a consistent integrated model out of the two updated models. As we show in Section 5, a simple way of approaching the problem is to use the forward and backward propagation operations, described in Section 3.2 both to detect conflicts and to implement the synchronization procedure. We say that this kind of approaches are *propagation-based*.

In general, the result of concurrent synchronization is not unique: each possible way of solving the existing conflicts may be considered a possible solution, where the users may decide whether each result corresponds to their needs. In particular, given a parallel update $\overline{u} = \langle u^S, u^T \rangle$, if there are conflicts between u^S and u^T, we may consider that each possible result corresponds to the merging of two conflict-free submodifications, $u_0^S \trianglelefteq u^S$ and $u_0^T \trianglelefteq u^T$, obtained after backtracking some of the conflicting operations involved in u^S and u^T. This condition is addressed in this section by the formal properties *soundness* and *maximal preservation*. For instance, in the example described in Section 2, if we have a source update that includes the deletion of an employee and a target update that includes a change on the salary of that employee, the two updates would be in conflict. To solve the conflict, we would need to either backtrack the deletion of the employee from the source model or to backtrack the change of salary of that employee in the target model.

Before studying the problem in more detail, we should note that if $\mathcal{M}_S$ and $\mathcal{M}_T$ are M-adhesive categories then the category of integrated models, $\mathcal{IM} = \mathcal{M}_S \times \mathcal{M}_T$, is also M-adhesive, and its updates are parallel updates $\overline{u} = \langle u^S, u^T \rangle$. Therefore we can apply the concepts and techniques presented in Section 3 to work with these updates.

More formally, we consider that a concurrent synchronization procedure *CSync* is a nondeterministic function whose input is a pair consisting of an integrated model M and a parallel update $\overline{u_1} : M \Rightarrow M_1$ and the output are parallel updates $\overline{u_2} : M \Rightarrow M_2$, where u_2^S and u_2^T could be seen as the result of merging u_1^S and u_1^T or some of its submodifications. Moreover, when we write $CSync(M,\overline{u_1}) = \overline{u_2}$ we mean that $\overline{u_2}$ is a possible result of this synchronization. Our notion of concurrent synchronization fits the procedure described in [XSHT13]. However, in [HEEO12], the result of concurrent synchronization is defined in a slightly different way. In that paper, a result of synchronizing $\overline{u_1} : M \Rightarrow M_1$ is a parallel update $\overline{u_2} : M_1 \Rightarrow M_2$, such that $M_2 \in \mathcal{C}$. From

a practical point of view, the latter notion suggests that, when we start the synchronization operation, the given updates u_1^S and u_1^T have already been performed, i.e. the given state consists of the models M_1^S and M_1^T, so the resulting updates u_2^S and u_2^T are applied to that state yielding the final state $\langle M_2^S, M_2^T \rangle$. This means that, if conflicts are found in the synchronization process, then u_2^S and u_2^T would have to undo some of the modifications included in u_1^S and u_1^T. On the contrary, the current notion suggests that, when the synchronization operation starts, the given updates u_1^S and u_1^T would have not been executed so the current state is still $\langle M^S, M^T \rangle$. Instead, these updates would have been kept or stored in some way, and the synchronization process would, first, find the possible conflicts between u_1^S and u_1^T, then, it would compute the resulting updates u_2^S and u_2^T and, finally, it would execute these updates over the initial state, leading to the final state $\langle M_2^S, M_2^T \rangle$. However, from a theoretical point of view, both notions can be considered equivalent, in the sense that, in general, from one kind of solution we can construct the other one.

Let us now see what properties should be satisfied by a propagation-based concurrent synchronization procedure *CSync* when applied to an integrated model M and a parallel update $\overline{u_1}$. The first three properties are just the concurrent version of the corresponding properties defined for propagation operations presented in Section 3.2. The first property states that any result must be consistent; the second one says that if the given modifications are the identity then the resulting updates should also be the identity (when the given interrelated model is consistent: otherwise the former properties would be contradictory); finally the third property is the concurrent version of hippocraticness. It just says that if the interrelated model after applying two updates is already consistent, then we may consider that the updates are already synchronized.

1. **Consistency:** Given the integrated model M and a parallel update $\overline{u_1} : M \Rightarrow M_1$, if $CSync(M, \overline{u_1}) = (\overline{u_2} : M \Rightarrow M_2)$, then $M_2 \in \mathcal{C}$.
2. **Identity:** If M is consistent and $CSync(M, \langle Id^S, Id^T \rangle) = u_2$ then $u_2 = \langle Id^S, Id^T \rangle$.
3. **Hippocraticness:** Given $\overline{u} : M \Rightarrow M_1$, if $M_1 \in \mathcal{C}$ then $CSync(M, \overline{u}) = \overline{u}$.

The above properties say very little about the relation between the resulting modifications and the given updates. In particular, a concurrent synchronization procedure that returns some updates that have nothing in common with the original modifications may satisfy these properties. Hence, we have to relate the output modifications with the input updates. In the context of propagation-based concurrent synchronization, we may consider that each update u^S on the source model not only specifies some given modifications on M^S, but it also specifies the modifications included in its propagation $fPpg(M, u^S)$ on the target model, and similarly for target updates. Therefore, in this framework, we may consider that a concurrent synchronization *CSync* is *sound* if whenever $\overline{u_2} = CSync(M, \overline{u_1})$, then all the modifications included in $\overline{u_2}$ are part of u_1^S and u_1^T or their propagation. This is stated by saying that the resulting updates can be obtained by merging some submodifications of the input updates and their propagation.

4. **Soundness:** If $CSync(M, \overline{u_1}) = \overline{u_2}$ then there are submodifications $\overline{v_1}$ and $\overline{v_2}$, such that $\overline{v_1} \trianglelefteq \overline{u_1}$, $v_2^S \trianglelefteq bPpg(M, u_1^T)$, $v_2^T \trianglelefteq fPpg(M, u_1^S)$, and $\overline{v_1} \otimes \overline{v_2} = \overline{u_2}$.

However, this is not enough. A concurrent synchronization procedure that always returns the identity updates would be considered sound. Obviously, this is not what we want. What we

may expect is that any resulting update $\overline{u_2}$ should include as much as possible the modifications specified by the input update $\overline{u_1}$. In particular, when there are no conflicts between u_1^S and u_1^T, $\overline{u_2}$ should include all the modifications specified by u_1^S and u_1^T.

5. **Maximal preservation:** If $CSync(M,\overline{u_1}) = \overline{u_2}$, then for all parallel updates $\overline{v_1} : M \Rightarrow M_1'$ and $\overline{v_2} : M \Rightarrow M_2'$, such that $\overline{v_1}$ is conflict-free, $\overline{v_1} \trianglelefteq \overline{u_1}$, $\overline{v_2} = \overline{v_1} \otimes \langle bPpg(M,v_1^T), fPpg(M,v_1^S)\rangle$ and $M_2' \in \mathcal{C}$, then if $\overline{u_2} \trianglelefteq \overline{v_2}$, we have $\overline{v_2} = \overline{u_2}$.

Finally, we may consider that, after conflict resolution, we have backtracked certain modifications included in the given input update $\overline{u_1}$, so that what we really want is to synchronize some conflict-free submodification $\overline{u_0} \trianglelefteq \overline{u_1}$. Then, we may also consider that the resulting update $\overline{u_2}$ should only include modifications from u_0^S and u_0^T and their propagation.

6. **Strong soundness:** If $CSync(M,\overline{u_1}) = \overline{u_2}$ then there is a parallel update $\overline{u_0} \trianglelefteq \overline{u_1}$, such that $u_0^S \otimes bPpg(M,u_0^T) = u_2^S$, and $fPpg(M,u_0^S) \otimes u_0^T = u_2^T$.

It is not difficult to see that strong soundness implies the properties of soundness and identity:

Proposition 1 *If $CSync(M,\overline{u_1}) = \overline{u_2}$ is a strongly sound solution then it also satisfies the properties of identity and soundness.*

Proof sketch.

- Identity. If $\overline{u_1}$ is the identity parallel update then, the only update $\overline{u_0}$ satisfying $\overline{u_0} \trianglelefteq \overline{u_1}$ is also the identity update. But, if $\overline{u_0}$ is the identity, according to the Identity property of basic synchronization, we have that $bPpg(M,u_0^T) = id$, and $fPpg(M,u_0^S) = id$. But this means that $u_0^S \otimes bPpg(M,u_0^T)$, and $fPpg(M,u_0^S) \otimes u_0^T$ are also the identity.
- Soundness. Let us suppose that $CSync(M,\overline{u_1}) = \overline{u_2}$ and there is an update $\overline{u_0}$, such that $\overline{u_0} \trianglelefteq \overline{u_1}$, $u_0^S \otimes bPpg(M,u_0^T) = u_2^S$, and $fPpg(M,u_0^S) \otimes u_0^T = u_2^T$. Then, by the property of preservation of decomposition of basic synchronization, we have that $bPpg(M,u_0^T) \trianglelefteq bPpg(M,u_2^T)$, and $fPpg(M,u_0^S) \trianglelefteq fPpg(M,u_2^S)$. Therefore, if we take as submodifications $\overline{v_1} = \overline{u_0}$, $v_2^S = bPpg(M,u_0^T)$, and $v_2^T = fPpg(M,u_0^S)$, then we have that $\overline{v_1}$ and $\overline{v_2}$ satisfy the conditions stated in the soundness property. ■

5 Strategies for propagation-based concurrent synchronization

In Section 4, we have seen some conditions that we may consider when reasoning about the adequacy of a given concurrent synchronization procedure. In this section we study different approaches to define propagation-based procedures for concurrent synchronization and we analyze up to which point they satisfy our correctness properties. In particular, in the first subsection we will study the conflict-free case: we will show how we can check the existence of conflicts between two updates u^S and u^T, and we will see that this check immediately tells us how to define the synchronization of these updates. Moreover, we will see that this procedure satisfies

all the properties defined in Section 4, except hippocraticness. Then, in the second subsection we consider the case where u^S and u^T are in conflict and we show that using propagation and conflict resolution, as presented in Section 3, we can build solutions that satisfy some of the above properties, but not all of them.

5.1 Conflict-free concurrent synchronization

The simplest approach to make conflicts explicit in a parallel update $\overline{u_1}$ is to consider simultaneously the given updates, u_1^S and u_1^T, and their propagation over the given integrated model. In particular, if we define $\overline{u_2} = \langle bPpg(M,u_1^T), fPpg(M,u_1^S)\rangle$, we can first check if $\overline{u_1}$ and $\overline{u_2}$ are in conflict, according to the notion of conflict studied in Section 3. However, this check only tells us if all modifications defined by $\overline{u_1}$ are compatible with the modifications defined by $\overline{u_2}$, but it does not tell us if the result of merging $\overline{u_1}$ and $\overline{u_2}$, $\overline{u_1} \otimes \overline{u_2}$, is consistent. In particular, if it is inconsistent, we should consider that there is also some kind of conflict between $\overline{u_1}$ and $\overline{u_2}$. Hence, a simple procedure to check conflict-freeness of $\overline{u_1}$ would be:

1. Let $\overline{u_2} = \langle bPpg(M,u_1^T), fPpg(M,u_1^S)\rangle$.
2. If $\overline{u_1}$ and $\overline{u_2}$ are conflict-free and $\overline{u_1} \otimes \overline{u_2}\colon M \to M_2$ with $M_2 \in \mathcal{C}$ then return **true**; otherwise return **false**.

Now, if $\overline{u_1}$ satisfies the above check, then we can define $CSync_0(M,\overline{u_1}) = (\overline{u_1} \otimes \overline{u_2})$ as the concurrent synchronization of $\overline{u_1}$.

Theorem 1 (Properties of conflict-free concurrent synchronization) *If $\overline{u_1} = \langle u_1^S, u_1^T\rangle$ is a conflict-free parallel update over M, then $CSync_0(M,\overline{u_1}) = (\overline{u_1} \otimes \overline{u_2})$, where $\overline{u_2} = \langle bPpg(M,u_1^T), fPpg(M,u_1^S)\rangle$, satisfies the properties of 1) Consistency, 2) Identity, 4) Soundness, 5) Maximal preservation and 6) Strong soundness.*

Proof sketch. According to Prop 1, strong soundness implies identity and soundness. Hence, it is enough to prove the properties of consistency, maximal preservation and strong soundness.

- Consistency. By construction, the result of $\overline{u_1} \otimes \overline{u_2}$ is consistent.
- Maximal preservation. If we have that $\overline{v_1} : M \Rightarrow M_1'$ and $\overline{v_1} \trianglelefteq \overline{u_1}$ this means that $\langle bPpg(M,v_1^T), fPpg(M,v_1^S)\rangle \trianglelefteq \langle bPpg(M,u_1^T), fPpg(M,u_1^S)\rangle$, as a consequence of the compatibility of propagation and update decomposition. But this means that $\overline{v_2} = \overline{v_1} \otimes \langle bPpg(M,v_1^T), fPpg(M,v_1^S)\rangle \trianglelefteq u_2$. Therefore, if $\overline{u_2} \trianglelefteq \overline{v_2}$, we have $\overline{v_2} = \overline{u_2}$.
- Strong Soundness. It is enough to take as submodification $\overline{u_0} = \overline{u_1}$. ■

Unfortunately, in general, $CSync_0$ does not satisfy hippocraticness. The reason is that, given $\overline{u_1} : M \Rightarrow M_1$, if $M_1 \in \mathcal{C}$, in general, we cannot expect that $\overline{u_1} = (\overline{u_1} \otimes \overline{u_2})$, where $\overline{u_2}$ is defined as above. Obviously, we can avoid this problem by checking if M_1 is consistent before starting the whole procedure. However, the rationale of hippocraticness is that synchronization or propagation procedures should perform the least amount of modifications to produce a consistent

integrated model. Then, by including this additional check, we avoid the extreme case, where no additional modifications are needed. However, from an informal viewpoint, we may conclude that propagation-based synchronization approaches are not hippocratic since, when propagating the given updates, we may be performing more modifications than needed.

In [XSHT13] a conflict-free concurrent synchronization procedure, called *SYNC*, is defined in a slightly different way. However, this difference causes that *SYNC* may fail to deliver a result in cases when there are no conflicts. Roughly[2], given a model M and a parallel update $\overline{u_1}$, *SYNC* builds the resulting update $\overline{u_2}$ in four steps:

- First, it does a backward propagation of u_1^T.
- Then, if there are no conflicts between u_1^S and $bPpg(M,u_1^T)$, it merges the two updates obtaining the source update $u_2^S = u_1^S \otimes bPpg(M,u_1^T)$.
- In the third step, it forward-propagates u_2^S, obtaining the target update $u_2^T = fPpg(M,u_2^S)$.
- Finally, it checks if the solution found, $\overline{u_2}$ *preserves* the given update $\overline{u_1}$, which essentially means that $\overline{u_2}$ includes all the modifications specified by $\overline{u_1}$. If $\overline{u_2}$ preserves $\overline{u_1}$, the procedure returns $\overline{u_2}$, otherwise it reports an error.

The problem can be seen in the following example. Let us suppose, following Section 2, that the source update u_1^S consists of the addition of some employee e_1 and the target update u_1^T consists of the addition of a different employee e_2 that works in the Berlin branch. Obviously, there is no conflict between the two updates. Now, according to [XSHT13], *SYNC* would behave as follows:

- The backward propagation of u_1^T would be the trivial identity update, since e_2 is not working in the Barcelona branch. As a consequence, $u_2^S = u_1^S$.
- $u_2^T = fPpg(M,u_2^S)$, i.e. u_2^T consists just of the addition of e_1 to the target model.
- Unfortunately, $\overline{u_2}$ does not preserve $\overline{u_1}$, since employee e_2 is not added now to the target model. Thus the procedure would deliver an error.

The problem is related with the properties of invertibility and strong invertibility of (backward) propagation. In particular, if $bPpg$ is strongly invertible then, if there are no conflicts, *SYNC* is always preserving, but if it is not strongly invertible then *SYNC* may be not preserving and, as a consequence, it may report an error, when the updates are conflict-free, as shown with the previous example.

Proposition 2 *If u_1^S and u_1^T are conflict-free updates of an integrated model M, $bPpg$ is strongly invertible, and $SYNC(M,\overline{u_1}) = \overline{u_2}$ then $\overline{u_1} \trianglelefteq \overline{u_2}$.*

Proof sketch. We know that $u_2^S = u_1^S \otimes bPpg(M,u_1^T)$, so by construction (cf. Section 3.1) $u_1^S \trianglelefteq u_2^S$. Let us now show that $u_1^T \trianglelefteq u_2^T$. By construction, we know that there is some target update v_1^T such

[2] The approach in [XSHT13] is not δ-based, but state-based. As a consequence, their formulation is slightly different.

that $v_1^T \circ bPpg(M,u_1^T) \stackrel{d}{=} u_2^S$. By the property of compatibility of propagation with update decomposition, we have that $fPpg(M',v_1^T) \circ fPpg(M,bPpg(M,u_1^T)) \stackrel{d}{=} fPpg(u_2^S)$, where M' is the integrated model obtained after applying the parallel update $\langle bPpg(M,u_1^T), fPpg(M,bPpg(M,u_1^T))\rangle$ on M. But, if $bPpg$ is strongly invertible, $fPpg(M,bPpg(M,u_1^T)) = u_2^T$ implying $u_1^T \trianglelefteq u_2^T$. ■

5.2 Conflicts and concurrent synchronization

Given an integrated model M, if the given parallel update $\overline{u_1}$ includes some conflicts, to ensure the properties of consistency, identity, maximal preservation and strong soundness, we would need to find (maximal) conflict-free subupdates $\overline{v} \trianglelefteq \overline{u_1}$. Then, for each such $\overline{v}$, we could find a solution just computing $CSync(M,\overline{v})$ as described in the previous section. A simple solution could consist: a) in computing all subupdates $\overline{v} \trianglelefteq \overline{u_1}$; b) for each $\overline{v}$, checking if it is conflict-free; and, finally, c) returning the maximal subupdates found. Obviously, the problem of this kind of approach is that it may be computationally very costly. In what follows we study two approaches that are more efficient in general, although we may be unable to ensure some of the above properties.

The first approach was proposed in [HEEO12] and can be seen as a variation (including conflict resolution) of the procedure SYNC defined in [XSHT13] and analyzed above. The proposed procedure, which we will call $CSync_1$ can be described in four steps. We assume to have an integrated model M and a parallel update $\overline{u} : M \Rightarrow M_1$:

- In the first step, we compute the forward propagation of u^S. Now, we have two updates over M^T, $u^T : M^T \to M_1^T$ and $fPpg(M,u^S) : M^T \to M_0^T$.
- If there are no conflicts between $u^T : M^T \to M_1^T$ and $fPpg(M,u^S) : M^T \to M_0^T$ we just merge these updates, otherwise we find a solution to them. Hence, after this step we have computed a target update $v^T : M^T \to M_3^T$ that is equal to the merging of conflict-free subupdates $v_0^T \trianglelefteq u^T$ and $v_1^T \trianglelefteq fPpg(M,u^S)$. Now, we may notice that there must exist an update w^T such that $w^T \circ u^T$ is equivalent to v^T. The reason is that if w_0^T is the update such that $u^T \stackrel{d}{=} w_0^T \circ v_0^T$ and w_1^T is the update such that $v^T \stackrel{d}{=} w_1^T \circ v_0^T$, it is enough to define $w^T = w_1^T \circ (w_0^T)^{-1}$. Intuitively, w^T first undoes the modifications which are part of u^T but that are not part of v^T, and then it applies the additional modifications w_1^T.
- Next, we find a maximal consistent submodel M_2^T of the target model M_3^T. Obviously, if M_3^T is already consistent then $M_2^T = M_3^T$. Let now $w_1^T : M_3^T \to M_2^T$ be the update that deletes all the elements in M_3^T which are not in M_2^T.
- Finally, we apply backward propagation of the update $w_1^T \circ w^T$ to M_1, leading to a source model M_2^S. In this context the resulting integrated model after the concurrent synchronization would be $\langle M_2^S, M_2^T\rangle$, which means that, in our terms[3], $CSync_1(M,\overline{u}) = \langle bPpg(w_1^T \circ w^T) \circ u^S, w_1^T \circ v^T\rangle$.

[3] As mentioned in Sect. 4, given M and a parallel update $u : M \Rightarrow M_1$, in [HEEO12], it is assumed that the result of the operation of concurrent synchronization is a parallel update over M_1

It is not difficult to prove that $CSync_1$ satisfies the properties of consistency and identity. In particular, we know that the resulting model $M_2 = \langle M_2^S, M_2^T \rangle$ is consistent because of the consistency of backward propagation, since, by construction, M_2^T is consistent and M_2 is obtained applying backward propagation to the update that lead to M_2^T. The case of the Identity property is simpler. It is enough to notice that if the given parallel update $\overline{u} : M \Rightarrow M_1$ is the identity, then all the updates defined in the above procedure, by construction, are also the identity. As a consequence, we have:

Theorem 2 (Properties of $CSync_1$) *$CSync_1$ satisfies the properties of consistency and identity.*

However, $CSync_1$ is not sound. Let us consider the example presented in Section 2 that is depicted in Fig. 2. In that example, the source update u^S includes the deletion of an employee e_1 with a base salary of 5000 euro and a bonus of 3000 euro, and the target update u^T includes a change of the address of e_1:

- Suppose, that the conflict between the deletion of e_1 and its change of address is solved delivering a target update w^T that does not include that deletion, and suppose that the resulting target model M_3^T is consistent.
- Now, e_1 is included in M_3^T with an overall salary of 8000 euros, but it is not included in M_1^S. Then, the backward propagation of w^T would include the addition of e_1 to the source model including a base salary of 6000 euro and a bonus of 2000 euro.

This modification of base salary and bonus is not specified by $\overline{u}$, so the synchronization is unsound.

The problem with soundness is, in a way, similar to the problem that we found in the previous section that causes that the procedure in [XSHT13] may be unable to find a synchronization, even if the given parallel update is conflict-free. However, in this case, if forward propagation is strongly invertible this does not necessarily mean that $CSync_1$ should be sound. The problem is in the third step, when constructing the maximal consistent submodel M_2^T of M_3^T we may delete some elements from the latter model that need not be deleted according to the given update.

The second approach, which we will call $CSync_2$, is an extension, including conflict resolution, of the procedure presented in Section 5.1. Again, we start assuming that we have a parallel update $\overline{u_1} : M \Rightarrow M_1$ over an integrated model M.

- First, we compute the parallel update $\overline{u_2} = \langle fPpg(M, u_1^S), bPpg(M, u_1^T) \rangle : M \Rightarrow M_2$.
- If $\overline{u_1}$ and $\overline{u_2}$ are conflict-free and $\overline{u_1} \otimes \overline{u_2} \in \mathcal{C}$, then we can define the result $\overline{u} = CSync(M, \overline{u_1}) = (\overline{u_1} \otimes \overline{u_2})$, as in the previous section. Otherwise, we look for a maximal update $\overline{u} : M \Rightarrow M'$ that is a solution to the conflicts and such that M' is consistent. How we can find $\overline{u}$ and M' depends on the specific framework. In particular, in the case of working with triple graph grammars, we could use a technique similar to consistency creation (see, e.g., [HEEO12]).

$CSync_2$ satisfies the properties of consistency, identity, soundness and maximal preservation. However, the procedure is not strongly sound, as the example below shows:

- Suppose that the source update u_1^S consists of the addition of a new employee e_1, working at a new department D to M^S. So the propagation of this update would include adding e_1 and D to the target model M^T.
- Suppose that adding e_1 to M^T creates a conflict with the given target update u_1^T (for example, because u_1^T includes the addition of another employee with the same name and it is forbidden to have in our model two employees with the same name).
- Suppose that the conflict is solved by not adding e_1 to M^T, which means that we would backtrack the addition of e_1 to M^S.
- Now, the fact that adding e_1 to the M^T creates a conflict, it does not mean that the addition of D creates any conflict. Hence, in the conflict resolution step we would keep the addition of D to M^T. However, if we have backtracked the addition of e_1 to M^S, adding D to M^T would not be a consequence of the propagation of the resulting source update, nor of the original target update. Hence, the procedure would not be strongly sound.

Theorem 3 *$CSync_2$ satisfies the properties of consistency, identity, soundness and maximal preservation.*

Proof sketch. $CSync_2$ is consistent since the resulting integrated model is consistent by construction. If $\overline{u_1}$ is the identity and M is consistent, then $\overline{u_1}$ has no conflicts and, as we have seen in Thm 1, the result of synchronization is the identity. $CSync_2$ is sound since, by construction, all the modifications included in $\overline{u_1}$ are, by construction, part of u_1 or of u_2. Finally, $CSync_2$ also satisfies maximal preservation because of the construction of $\overline{u_1}$. ■

6 Related Work

Incremental model transformation or (standard) model synchronization is a problem that has been largely studied in different areas of computer science like databases, software engineering, and programming languages (see, e.g., [DB82, FKPT08, Ste10, HPW11, BPV06, TCC12]. However, to our knowledge, the only approaches that deal with the problem of concurrent synchronization of general models were presented by Xiong et al. [XSHT09, XSHT13] handling the conflict free case and by Hermann et al. [HEEO12] handling the general case including possibly conflicting updates. We studied both of them in detail in this paper based on an abstract framework of for propagation-based synchronization. Other authors have also considered the problem of concurrent synchronization, albeit in a more restrictive setting. In particular, Foster et al. [FGK+05] consider this problem, but models are restricted to tree like structures and the target model is an abstract of the source. Also, Xiong et al.[XHZ+09] consider this problem when updates are defined in terms of a given set of operations.

Different kinds of synchronization frameworks that are based on the concept of lenses [BPV06, HPW12] solve the view update problem in the domain of programming languages. Some of these works considered the put-put law as a major requirement [DXC11a, GJ12], which ensures compositionality in a rigorous manner. However, as discussed in several

papers and also for our example, the put-put law is often too restrictive for solving the general concurrent synchronization problem. Therefore, we presented the properties of soundness and strong soundness that do not enforce compositionality in general, but only in cases where composition of the given updates is compatible with decomposition. This allows us to show that concurrent synchronization can ensure soundness (Theorem 3) and even strong soundness in the case of conflict free updates (Theorem 1).

However, propagation-based approaches have also some disadvantages. The main one, which is not explicit in the paper, is that these procedures may be unable to find some reasonable solutions for the synchronization of a given parallel update. The problem is caused by the fact that, given an update on a given model (e.g. the source model), there may be different ways of propagating that update to the target model. However, since propagation operations are supposed to be deterministic, the operation $fPpg$ would only implement one of these ways. Hence, given a parallel update u, the solutions that we may obtain by a propagation-based procedure would only be built from the specific ways of propagating u^S and u^T implemented in $fPpg$ and $bPpg$. In particular, if $fPpg$ and $bPpg$ differ from each other in this sense, then the derived synchronization approaches are not hippocratic [Ste08], which would require that the synchronization has no effect, if the current integrated model is already consistent. The second problem is related with the fact that, in case of conflicts, the procedures that we have studied do not satisfy some of the properties (especially, soundness or strong soundness) that, a priori, we thought that they should satisfy. However, this is not as important as it may seem to be. Our soundness properties are based on the idea that the modifications specified by each source or target update consist of the modifications included in the update and the ones included in its propagation. We believe that this is reasonable in the context of propagation-based methods, but it would be too strong in general. The reason is that, according to the ideas discussed above, there may be reasonable solutions to the synchronization problem that are not based on the given propagation operations. As a consequence, these solutions would probably be unsound, in view of our notion of soundness.

7 Conclusion and Future Work

In this paper, we provided a general and abstract framework for handling the problem of concurrent model synchronization using propagation-based approaches. The use of propagation operations has mainly two advantages. The first one is that they provide a simple way of checking the existence of conflicts in a given parallel update. The second one is that the operations for concurrent synchronization are relatively easy to implement, since we may reuse propagation operations defined for the basic synchronization problem. In our main results (Theorem 1 and Theorem 3), we have shown under which conditions propagation based synchronization approaches ensure maximal preservation of the given updates and soundness (avoidance of side effects).

For further work, we believe that it will be interesting to define a concurrent synchronization procedure that is not propagation-based, studying its feasibility and correctness.

Bibliography

[BPV06] A. Bohannon, B. C. Pierce, J. A. Vaughan. Relational lenses: a language for updatable views. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, 2006.

[DB82] U. Dayal, P. A. Bernstein. On the Correct Translation of Update Operations on Relational Views. *ACM Trans. Database Syst.* 7(3):381–416, 1982.

[DXC10] Z. Diskin, Y. Xiong, K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations. In *Proc. ICMT 2010*. Lecture Notes in Computer Science 6142, pp. 61–76. Springer, 2010.

[DXC11a] Z. Diskin, Y. Xiong, K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *Journal of Object Technology* 10:6: 1–25, 2011.

[DXC+11b] Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, F. Orejas. From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case. In *Model Driven Engineering Languages and Systems, MODELS 2011*. Lecture Notes in Computer Science 6981, pp. 304–318. Springer, 2011.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs of Theoretical Comp. Sc. Springer, 2006.

[EET11] H. Ehrig, C. Ermel, G. Taentzer. A Formal Resolution Strategy for Operation-Based Conflicts in Model Versioning Using Graph Modifications. In *Fundamental Approaches to Software Engineering, FASE 2011*. Lecture Notes in Computer Science 6603, pp. 202–216. Springer, 2011.

[EGH10] H. Ehrig, U. Golas, F. Hermann. Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. *Bulletin of the EATCS* 102:111–121, 2010.

[FGK+05] J. N. Foster, M. B. Greenwald, C. Kirkegaard, B. C. Pierce, A. Schmitt. Schema-directed data synchronization. Technical report MS-CIS-05-02, University of Pennsylvania, 2005.

[FKPT08] R. Fagin, P. G. Kolaitis, L. Popa, W. C. Tan. Quasi-inverses of schema mappings. *ACM Trans. Database Syst.* 33(2), 2008.

[GJ12] J. Gibbons, M. Johnson. Relating Algebraic and Coalgebraic Descriptions of Lenses. *Electronic Communications of the EASST (ECEASST)* 49, 2012.

[HEEO12] F. Hermann, H. Ehrig, C. Ermel, F. Orejas. Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars. In *Fundamental Approaches to Software Engineering, FASE 2012*. Lecture Notes in Computer Science 7212, pp. 178–193. Springer, 2012.

[HEO+11] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, Y. Xiong. Correctness of Model Synchronization Based on Triple Graph Grammars. In *Model Driven Engineering Languages and Systems, MODELS 2011*. Lecture Notes in Computer Science 6981, pp. 668–682. Springer, 2011.

[HPW11] M. Hofmann, B. C. Pierce, D. Wagner. Symmetric lenses. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*. Pp. 371–384. ACM, 2011.

[HPW12] M. Hofmann, B. C. Pierce, D. Wagner. Edit lenses. In Field and Hicks (eds.), *Proc. Symp. on Principles of Programming Languages (POPL'12)*. Pp. 495–508. ACM, 2012.

[LS05] S. Lack, P. Sobocinski. Adhesive and quasiadhesive categories. *Theor. Inf. App.* 39:511–545, 2005.

[SK08] A. Schürr, F. Klar. 15 Years of Triple Graph Grammars. In *Proc. Int. Conf. on Graph Transformation (ICGT 2008)*. Pp. 411–425. 2008. doi:10.1007/978-3-540-87405-8_28

[Ste08] P. Stevens. Towards an Algebraic Theory of Bidirectional Transformations. In Ehrig et al. (eds.), *Proc. Int. Conf. on Graph Transformation (ICGT'08)*. Lecture Notes in Computer Science 5214, pp. 1–17. Springer, 2008.

[Ste10] P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and System Modeling* 9(1):7–20, 2010.

[Ste12] P. Stevens. Observations relating to the equivalences induced on model sets by bidirectional transformations. *ECEASST* 49, 2012.

[TCC12] J. F. Terwilliger, A. Cleve, C. Curino. How Clean Is Your Sandbox? - Towards a Unified Theoretical Framework for Incremental Bidirectional Transformations. In *Theory and Practice of Model Transformations - 5th International Conference, ICMT 2012*. Lecture Notes in Computer Science 7307, pp. 1–23. Springer, 2012.

[XHZ+09] Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi, H. Mei. Supporting automatic model inconsistency fixing. In *ESEC/FSE 2009*. Pp. 315–324. 2009.

[XSHT09] Y. Xiong, H. Song, Z. Hu, M. Takeichi. Supporting Parallel Updates with Bidirectional Model Transformations. In *Theory and Practice of Model Transformations, ICMT 2009*. Lecture Notes in Computer Science 5563, pp. 213–228. Springer, 2009.

[XSHT13] Y. Xiong, H. Song, Z. Hu, M. Takeichi. Synchronizing concurrent model updates based on bidirectional transformation. *Software and System Modeling* 12(1):89–104, 2013.