Technische Universität Berlin

Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Correctness of Generalisation and Customisation of Concurrent Model Synchronisation Based on Triple Graph Grammars

Susann Gottmann, Frank Hermann, Nico Nachtigall, Benjamin Braatz,
Claudia Ermel, Hartmut Ehrig, and Thomas Engel

Bericht-Nr. 2013-08
ISSN 1436-9915

Correctness of Generalisation and Customisation of Concurrent Model Synchronisation Based on Triple Graph Grammars

Susann Gottmann
Interdisciplinary Centre for Security, Reliability and Trust
Université du Luxembourg
Luxemburg
susann.gottmann@uni.lu

Frank Hermann
Interdisciplinary Centre for Security, Reliability and Trust
Université du Luxembourg
Luxemburg
frank.hermann@uni.lu

Nico Nachtigall
Interdisciplinary Centre for Security, Reliability and Trust
Université du Luxembourg
Luxemburg
nico.nachtigall@uni.lu

Benjamin Braatz
Interdisciplinary Centre for Security, Reliability and Trust
Université du Luxembourg
Luxemburg
benjamin.braatz@uni.lu

Claudia Ermel
Technische Universität Berlin
Germany
claudia.ermel@tu-berlin.de

Hartmut Ehrig
Technische Universität Berlin
Germany
hartmut.ehrig@tu-berlin.de

Thomas Engel
Interdisciplinary Centre for Security, Reliability and Trust
Université du Luxembourg
Luxemburg
thomas.engel@uni.lu

Triple graph grammars (TGGs) have been successfully applied to specify and analyse bidirectional model transformations. Recently, a formal approach to concurrent model synchronisation has been presented, where a source and a target modification have to be synchronised simultaneously. In this approach, conflicts between the given and propagated source or target model modifications are taken into account. A semi-automatic conflict resolution strategy is proposed, where a formal resolution strategy can be combined with a user-specific strategy. Up to now, our approach requires deterministic propagation operations.

In this paper, we want to relax this condition and also consider non-deterministic (conflicting) operations which might require backtracking. For *optimisation*, we propose to eliminate conflicts between the operational rules of a TGG using the concept of *filter NACs*. Nevertheless, concurrent synchronisation is non-deterministic from a user perspective: The user may choose between *forward synchronisation* and *backward synchronisation*. Moreover, the conflict resolution strategy may result in several solutions from which the user has to select the most adequate one. Hence, we discuss different kinds of *customisation* of the synchronisation process and explain the impacts of the different strategies.

1 Introduction

Bidirectional model transformations have been successfully modelled and analysed using triple graph grammars (TGGs) [25, 26]. More recently, TGGs have also been applied in case studies for model integration and model synchronisation [20, 7, 9].

Model synchronisation aims to propagate updates between interrelated domains in order to derive updated models that are consistent with each other. Since model changes may occur concurrently in related domains and in a distributed way, model synchronisation has to cope with merging updates and resolving conflicts in the general case.

In order to perform model synchronisation of concurrent updates, we use the concurrent model synchronisation approach based on triple graph grammars [12]. In this approach, model synchronisation is performed by propagating the changes from one model of one domain to a corresponding model in another domain using forward and backward propagation operations. The propagated changes are compared with the given local updates. Possible conflicts are resolved in a semi-automated way. The operations are realised by model transformations based on TGGs [15, 18] and tentative merge constructions solving conflicts [6].

The synchronisation framework presented in [12] is based on deterministic propagation operations fPpg and bPpg. In a more general setting with non-deterministic operations, the procedure would require backtracking and yield several possible results. In this paper, we also consider non-deterministic (conflicting) operations which might require backtracking. We propose to eliminate conflicts between the operational rules of a TGG using the concept of *filter NACs* that was already successfully applied in the area of model transformations [13]. For this purpose, we extend certain synchronisation operations automatically with compatible NACs and show that these changes do not affect the correctness and completeness results for the derived synchronisation framework. This first main result of the paper ensures that concurrent model synchronisation can be performed efficiently also in cases where the TGG operations are initially not deterministic, but become deterministic via the applied extension for optimisation.

Nevertheless, concurrent synchronisation might be non-deterministic from a user perspective: The user may choose between *forward synchronisation* (where the changes on the source domain are propagated to the target domain) and *backward synchronisation* (where target domain changes are propagated back to the source domain). Moreover, the conflict resolution strategy may result in several solutions from which the user has to select the most adequate one. User input is required to guide the synchronisation process. Hence, in the second contribution in this paper, we discuss different kinds of *customisation* of the synchronisation process and explain the impacts of the different customisation strategies.

The paper is structured as follows: We present our running example in Sec. 2, a concurrent model synchronisation problem in the area of business modelling. Sec. 3 reviews the main concepts and results of the formal framework for concurrent model synchronisation [15, 12, 18]. In Sec. 4, we generalise the formal framework to non-deterministic forward and backward propagation operations and describe our new conflict elimination strategy based on filter NACs. Sec. 5 and Sec. 6 compare different customisation strategies to guide the user in obtaining a desired synchronisation result. After discussing related work in Sec. 7, we conclude the paper with an outlook to future work in Sec. 8.

2 Application Scenario

In this section, our running example will be introduced in presenting the concurrent model synchronization problem (Sec. 2.1). We will distinguish different kinds of model updates, which cause different effects during the synchronization process. The corresponding TGG used for the derivation of the corresponding synchronisation framework is provided in Sec. 3 thereafter.

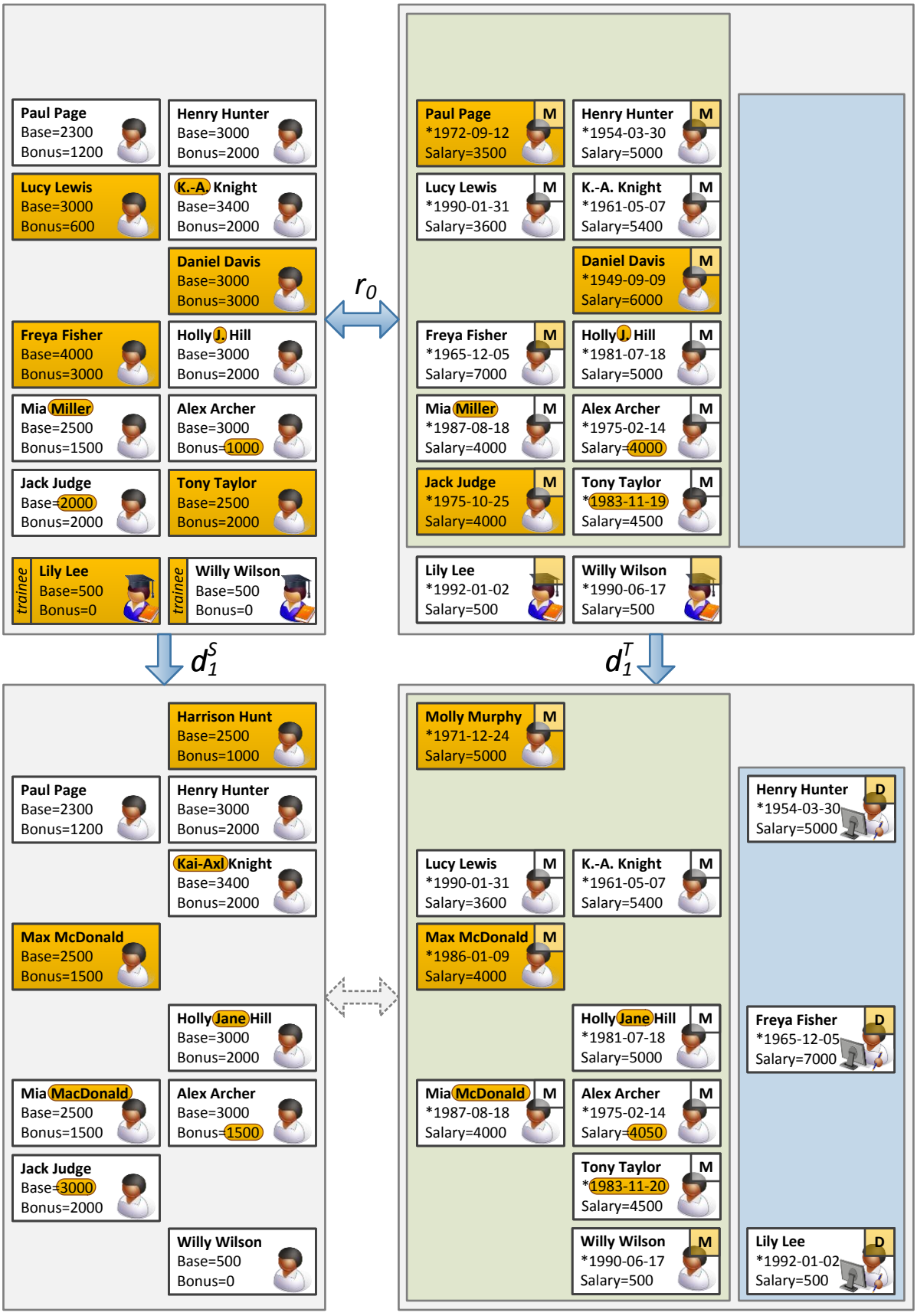


Figure 1: Concurrent model synchronization: compact example

2.1 Concurrent model synchronization problem

In the example shown in Fig. 1, we depict two models that are assumed to be in correspondence. As we may see, each model stores information about the employees of a company, where the information in one model is slightly different from the information on the other model. Moreover, the source model (on the left side of the visualization in Fig. 1) is assumed to include information only about the employees from the marketing department and the trainees of the company who do not belong to any department, while the employees in the target model (on the right side of the visualization in Fig. 1) may belong to any department or to no department, i.e., they are trainees. We assume that the source domain represents the staff administration information available to the secretary of the marketing department whereas the target domain contains information for the human resource manager or the administration department of the whole company, respectively.

The source model provides detailed salary information that is divided in the Base and Bonus for each person. Furthermore in the source domain, the distinction between a trainee and an employee of the marketing department is indicated by a flag isTrainee. If this flag is set to true, the person is a trainee, if it is set to false, it is an employee of the marketing department. The type graph of the abstract syntax representation of the source model explicitly contains this flag and is described in detail in Sec. 3. In Fig. 1, this distinction is illustrated by the bar trainee on the left side of the box containing the data of the person. If the bar trainee is available, the person is a trainee, otherwise he is an employee of the marketing department. Furthermore, the role icon for a trainee is wearing a hat and carrying book.

The target domain information regarding the complete staff of the company is available. It contains information of the whole Salary and the date of birth of each employee. Furthermore, the membership of each employee to a department is indicated. A person does not belong to any department, if he is a trainee. In Fig. 1, all available departments in the target domain are indicated by different coloured boxes including all employees of that department. In addition a small letter on the upper right corner of each box belonging to a person indicates the corresponding department and also a special role icon. In our example, the green box and letter "M" refers to employees of the marketing department. The blue box and the letter "D" denotes people employed at the development department. The role icon of employees of the development department shows a person carrying a pencil and working in front of a monitor. If a person is outside of any coloured box and if there is no letter available, the person is a trainee. In the target domain, the rule icon of a trainee is identical to the one in the source domain. E.g. in our example given in Fig. 1, Tony Taylor is an employee of the marketing department visualized by the letter "M" and because his data record is assigned to the marketing department, i.e., the green box. In contrast, Lily Lee is a trainee because she does not belong to any department visualized by a missing letter and her data record box is not part of any coloured box referring to any department.

Two model updates have to be synchronized concurrently: On the source side model update d_1^S and on the target side model update d_1^T will be performed as shown in concrete syntax in Fig. 1. There are different kinds of updates which can be performed on each side representing specific actions for the corresponding administrative personnel. On both sides, a person can be removed because he/she terminated his/her work in the company, e.g., because the person retires. The opposite, hiring a new employee, i.e., adding a new person, is possible on each side. Furthermore, attributes of a data record can be changed, e.g., the name of a person, the salary, etc. and a person can be transferred from a department to another department or added to a department, i.e. a trainee can be hired by a department. These actions can be performed in parallel and on both sides of the model (source and target domain) so that conflicts might occur, e.g., because the same data record is changed differently on both sides. In our application scenario we perform various modifications concurrently. Consequently, conflicts might occur

which can sometimes be solved automatically, but in some cases conflict resolution requires a manual intervention by the user. In the following all modifications of our application scenario will be presented. In Sec. 5 the propagation of model updates from one domain into the other domain is presented and compared with the propagation into the opposite direction. We will recognise that a different order of the propagation will cause different results or different conflicts, respectively. We will investigate both kinds of conflict resolution: the automatic conflict resolution as well as the manual conflict resolution, in both possible propagation directions in Sec. 6.

All model updates on the source and target domain of the application scenario visualized in Fig. 1 can be divided into two kinds of model updates. Both model updates and the corresponding cases of our scenario are presented in detail in the following Sec. 2.1.1 and Sec. 2.1.2, respectively. The two kinds of model updates are:

1. Model updates on one domain only, which are presented in table 1. If a model update will be performed for a given data record in one domain only, the update will be directly propagated to the corresponding data record in the other domain, because no conflicts will occur.
2. Concurrent model updates, which are listed in table 2. If a model update on a data record will be performed on both domains concurrently, conflicts might occur that will prevent an automated propagation of the modifications into the other domain.

2.1.1 Model Updates on one Domain

In table 1 all model updates are enumerated that take place on one domain only of our scenario, which is illustrated in Fig. 1. We distinguish between the following kinds of model updates on one domain:

- If an Addition is performed on one domain of the model, a new complete data record of an employee or trainee, respectively, will be created or an existing data record will be extended by an attribute or a new connection from an existing trainee to a department will be created, i.e., a previous trainee will be employed by a department.
- If a model update is a Deletion, complete data records, single attributes or the connection between an employee and a specific department will be deleted.
- The Deletion includes the Addition-Deletion update, which is presented in case 4, where a connection between an employee and a department will be deleted but a new connection from this employee to another department created, i.e., the employee will be redeployed by another department.
- The Attribute Modification includes all modifications of existing attributes, e.g., change name, birth date, salary, base or bonus. This kind of model update does not include the modification of the corresponding department, because in this special case, the change of the department is a Addition-Deletion update, as explained above.

Note that in table 1 and table 2 the abbreviated single types of model updates mean: “A” is Addition, “D” is Deletion, “M” is Attribute Modification, “DA” is Deletion-Addition and “-” is no changes.

Example 2.1 (Case 1 - Appointment of Molly Murphy)**.** In the target domain, Molly Murphy will be employed in the marketing department by the human resource manager of the whole company with model update d_1^T. Molly Murphy will not be employed by the marketing department itself, so that this model update will not be carried out in the source domain, therefore it is not part of the modification d_1^S. This case represents a model update only in the target domain of the kind Addition.

	Case	Source Model	Target Model	
Addition	1	No changes.	Add *Molly Murphy* to marketing department.	-,A
	2	Add *Harrison Hunt*.	No changes.	A,-
Deletion	3	No changes.	Remove *Paul Page*.	-,D
	4	No changes.	Redeploy *Henry Hunter* from marketing department in development department.	-,DA
	5	Remove *Lucy Lewis*.	No changes.	D,-
Attribute Modification	6	Change name *K.-A. Knight* to *Kai-Axl Knight*.	No changes.	M,-

Table 1: Summary of the model updates on one domain only in our application scenario.

Example 2.2 (Case 2 - Appointment of Harrison Hunt)**.** The marketing department itself employs Harrison Hunt, but not the human resource manager of the company. Consequently, this model update is reflected in the model updates d_1^S in the source domain, but not in d_1^T in the target domain. This case represents a model update only in the source domain of the kind Addition.

Example 2.3 (Case 3 - Resignation of Paul Page)**.** Paul Page wants to quit from his work at the marketing department. This update will be performed by the human resource manager on the target domain, but not by the administration of the marketing department. Consequently, this update is part of the modification d_1^T, but not of d_1^S. This case represents a model update only in the target domain of the kind Deletion.

Example 2.4 (Case 4 - Transfer of Henry Hunter)**.** Henry Hunter will leave the marketing department because he will switch to the development department. This redeployment will be entered by the human resource manager into the target domain, which is included in the modification d_1^T. Modification d_1^S does not reflect this update, because the administration of the marketing department will not enter this model update. This case represents a model update only in the target domain of the kind Addition-Deletion.

Example 2.5 (Case 5 - Resignation of Lucy Lewis)**.** Lucy Lewis wants to leave the company. This change is included in the modification d_1^S because it will be entered by the administration of the marketing department in the source domain. In contrast, on the target domain, this model update will not be performed so that d_1^T will not include this specific update. This case represents a model update only in the source domain of the kind Deletion.

Example 2.6 (Case 6 - Renaming of K.-A. Knight)**.** The employee K.-A. Knight is currently registered in both domains with the abbreviated first name *K.-A.*. The marketing department detects this mistake and corrects his first name to *Kai-Axl*, which is reflected in the modification d_1^S. In contrast, this change will not be performed on the target side by modification d_1^T. This case represents a model update only in the source domain of the kind Attribute Modification.

2.1.2 Concurrent Model Updates on Both Domains

table 2 enumerates all cases of our scenario where concurrent model updates on both domains are performed. We can distinguish between the following kinds of model updates on both domains, which are also considered in table 2:

- Addition vs. Addition: A concurrent addition on both domains of the model means that the same data record or attribute or edge indicating the membership to a department, respectively, will be

added concurrently. Even if the addition of the same data on both sides will not produce any conflicts, it can lead to an undesirable side effect: A model update adding the same thing on both domains will result in the creation of the same thing twice after applying the propagation operation. E.g. if a new employee is added on the source and on the target side, concurrently, the employee will be added twice (c.f. case 7).

- Deletion vs. Deletion: On both domains, the same data record, attribute or membership to a certain department, respectively, can be deleted concurrently that will always produce delete-delete conflicts. This kind of concurrent model update can also contain Deletion vs. Addition-Deletion. Then an element will be removed in one domain but modified in the other domain, which lead to delete-use conflicts.
- Attribute Modification vs. Attribute Modification: Attributes belonging to the same person (i.e., employee or trainee, respectively) are modified in both domains. When, in addition, the modifications d_1^T and d_1^S change attributes on source and target domain which correspond to each other, delete-delete conflicts will occur.
- Deletion vs. Attribute Modification: The next kind of model update deletes an object in one domain but modifies an attribute of the corresponding object in the other domain. This type of model update might provoke delete-use conflicts.
- Addition vs. Deletion or Attribute Modification: The last kind of model update consists of two similar types of model updates: Adding attributes or creating a membership to a department in one domain, whereas the data record will be deleted in the other domain and adding attributes or creating a membership to a department in one domain, whereas an attribute of the same object will be modified in the other domain. Depending on the correspondences between the elements affected by the addition in one domain and the deletion or attribute modification, respectively, in the other domain, conflicts might occur.

Example 2.7 (Case 7 - Appointment of Max McDonald)**.** Max McDonald will be hired as new employee of the marketing department. This model update will be performed on both sides, on the source domain by the administration of the marketing department and on the target side by the human resource manager, which will be reflected in both modifications d_1^S and d_1^T. This case represents a concurrent model update of the kind Addition vs. Addition.

Example 2.8 (Case 8 - Retirement of Daniel Davis)**.** Daniel Davis was employed at the marketing department. Due to reaching the retirement age, he will leave the company. The deletion of Daniel Davis will is performed concurrently in both domains at the same time. Consequently, the deletion of Daniel Davis is part of the modification d_1^S but also of d_1^T. This case represents a concurrent model update of the kind Deletion vs. Deletion.

Example 2.9 (Case 9 - Transfer of Freya Fisher)**.** Freya Fisher is currently employed at the marketing department but will be transferred to the development department. This change is entered on both sides of the model. In the source domain, Freya Fisher will be deleted because the source model only depicts employees of the marketing department or trainees, respectively. In contrast, in the target domain, the connection from Freya Fisher to the marketing department will be deleted, first, in order to create a new connection from Freya Fisher to the development department characterizing the new membership to the development department. This case represents a concurrent model update of the kind Deletion vs. Deletion-Addition. In this case of our example, the Deletion takes place in the source domain in d_1^S, whereas the Deletion-Addition model update takes place in the target domain in d_1^T.

	Case	Source Model	Target Model	
Addition	7	Add *Max McDonald*.	Add *Max McDonald* to marketing department.	A,A
Deletion	8	Remove *Daniel Davis*.	Remove *Daniel Davis*.	D,D
	9	Remove *Freya Fisher*.	Redeploy *Freya Fisher* from marketing department in development department.	D,DA
Attribute Modification	10	Change name *Holly J. Hill* to *Holly Jane Hill*.	Change name *Holly J. Hill* to *Holly Jane Hill*.	M,M
	11	Change name *Mia Miller* to *Mia MacDonald*.	Change name *Mia Miller* to *Mia McDonald*.	M,M
	12	Increase bonus of *Alex Archer* from 1000 to 1500.	Increase salary of *Alex Archer* from 4000 to 4050.	M,M
Deletion vs. Attribute Modification or Addition	13	Increase base of *Jack Judge* from 2000 to 3000.	Remove *Jack Judge*.	M,D
	14	Remove *Tony Taylor*.	Increase salary of *Tony Taylor* from 4500 to 5000.	D,M
Addition vs. Deletion or Attribute Modification	15	Remove trainee *Lily Lee*.	Employ *Lily Lee* at Development department.	D,A
	16	Employ *Willy Wilson* at Marketing department.	Employ *Willy Wilson* at Marketing department.	M,A

Table 2: Summary of model updates on both domains in our application scenario (concurrent changes).

Example 2.10 (Case 10 - Renaming of Holly J. Hill)**.** Holly J. Hill intimates her full first name. The change of the first name from Holly J. to Holly Jane will be entered on both domains and therefore will be reflected in the model updates d_1^S and d_1^T. This case represents a concurrent model update of the kind Attribute Modification vs. Attribute Modification.

Example 2.11 (Case 11 - Renaming of Mia Miller)**.** Due to marriage, the last name of Mia Miller changed to McDonald. In the target domain, this change is entered correctly, which is reflected in d_1^T. However, the administration of the marketing department enters this update with a typing error: In the source domain, the new last name is modified from Miller to MacDonald, which is reflected in modification d_1^S. This case represents a concurrent model update of the kind Attribute Modification vs. Attribute Modification.

Example 2.12 (Case 12 - Changing the salary and bonus of Alex Archer)**.** The Bonus of Alex Archer will be increased from *1000* to *1500* by the marketing department, because he was performing well during the last years. This update of Alex Archers' Bonus is entered by the administration of the marketing department on the source domain, which is part of model update d_1^S. Furthermore, the Salary of Alex Archer will be increased from *4000* to *4050* by the human resource manager of the whole company in the target domain in order to compensate the inflationary adjustment, which is reflected in modification d_1^T. Both model updates are independent from each other, therefore, they both should be carried out leading to a new total salary of *4550* and a new *Base = 3050* and a new *Bonus = 1500*, respectively. In Ex. 5.12 and Ex. 6.12 we will see that the model update regarding case 12 will produce delete-delete conflicts and will propose different possible results from the desired one. This case represents a concurrent model update of the kind Attribute Modification vs. Attribute Modification.

Example 2.13 (Case 13 - Resignation and changing the base salary of Jack Judge)**.** After being a long-time employee of the marketing department, Jack Judge wants to leave the company next month. This model update is entered by the human resource manager of the company in the target domain and will be executed as part of modification d_1^T. A rule that is specific to the marketing department states that the base salary will be increased every two years. Jack Judge has now reached this time so that his Base will be increased from *2000* to *3000* by the marketing department. This change is entered by the marketing department itself in the source domain (model update d_1^S). Both model updates are completely opposed to each other so that conflicts will occur requiring a manual resolution by the user. This case represents a concurrent model update of the kind Attribute Modification vs. Deletion.

Example 2.14 (Case 14 - Resignation and changing the birth date of Tony Taylor)**.** Tony Taylor, who has been an employee of the marketing department, will leave the company next month. The administration of the marketing department enters this change in the source domain, which will be reflected by modification d_1^S. In addition, another Tony Taylor will be hired for the marketing department. This leads to a misunderstanding by the human resource manager of the whole company, who modifies the data record of the old Tony Taylor in the target domain by mistakenly changing the birth date instead of adding a the new employee. The change in the target domain is included in modification d_1^T. In propagating both model updates d_1^S and d_1^T, conflicts will occur, which will be discussed later in Ex. 5.14 and Ex. 6.14. This case represents a concurrent model update of the kind Deletion vs. Attribute Modification.

Example 2.15 (Case 15 - Transfer of Lily Lee)**.** Lily Lee has been a trainee at the company and was hired for a permanent position by the development department. The change of her position will be entered concurrently in both domains. Her data record will be deleted in the source domain, because it only reflects information about employees of the marketing department or of trainees. In the target domain, the membership of Lily Lee to the development department is added. This case represents a concurrent model update of the kind Deletion vs. Addition.

Example 2.16 (Case 16 - Switch of Willy Wilson to a permanent position)**.** Willy Wilson has been a trainee at the marketing department was performing well so that he gets hired by the marketing department for a permanent position. This model update is entered on both domains. In the source domain, the attribute value of isTrainee is changed from true to false. In the target domain, the membership of Willy Wilson to the marketing department is added. This case represents a concurrent model update of the kind Attribute Modification vs. Addition.

Note, even if we address each modification separately as one case, all modifications on source and target side are combined to one model update $d_1 = d_1^S \cup d_1^T$, i.e., they are executed concurrently. We only address the cases separately for better clarity.

After performing updates d_2^S and d_2^T, a "consistently integrated model" (see below) should be derived that reflects as many changes as possible from the original updates d_1^S and d_1^T in both domains and resolves inconsistencies. E.g. in case 8, the deletion of Daniel Davis is entered in both domains of the model, which will cause a delete-delete conflict. But, we may notice that this conflict can be considered not a real conflict, but an apparent one, because the deletion can be propagated automatically. In contrast, in case 12, the Salary (in the target domain) and the Bonus (in the source domain) of Alex Archer will be increased, which will lead to a delete-delete conflict, too. But in this case, a manual intervention by the user is necessary in order to receive the desired results.

To solve conflicts like the ones mentioned in this section (Sec. 2.1.2), an adequate conflict resolution strategy must be integrated in the concurrent model synchronization process.

2.1.3 Expected Results

In Fig. 2 the desired optimal results of our scenario are illustrated. In Sec. 5 and Sec. 6, we will investigate the possible derivations of solutions for each single case in detail and also for the whole scenario in connection. At the end we will see, if we can derive the desired results in a consistently integrated model after performing the model updates d_2^S and d_2^T.

In the following, we would like to present the desired results for each case, separately, without taking into account that the consistently integrated model can only be derived in propagating all changes from source to target model, first (fSync), or the other way round (bSync). Depending on the synchronization direction chosen, i.e., if fSync should be performed or bSync, different results will emerge. Therefore, the desired results illustrated in Fig. 2 possibly might not be achieved (c.f. Sec. 5 and Sec. 6).

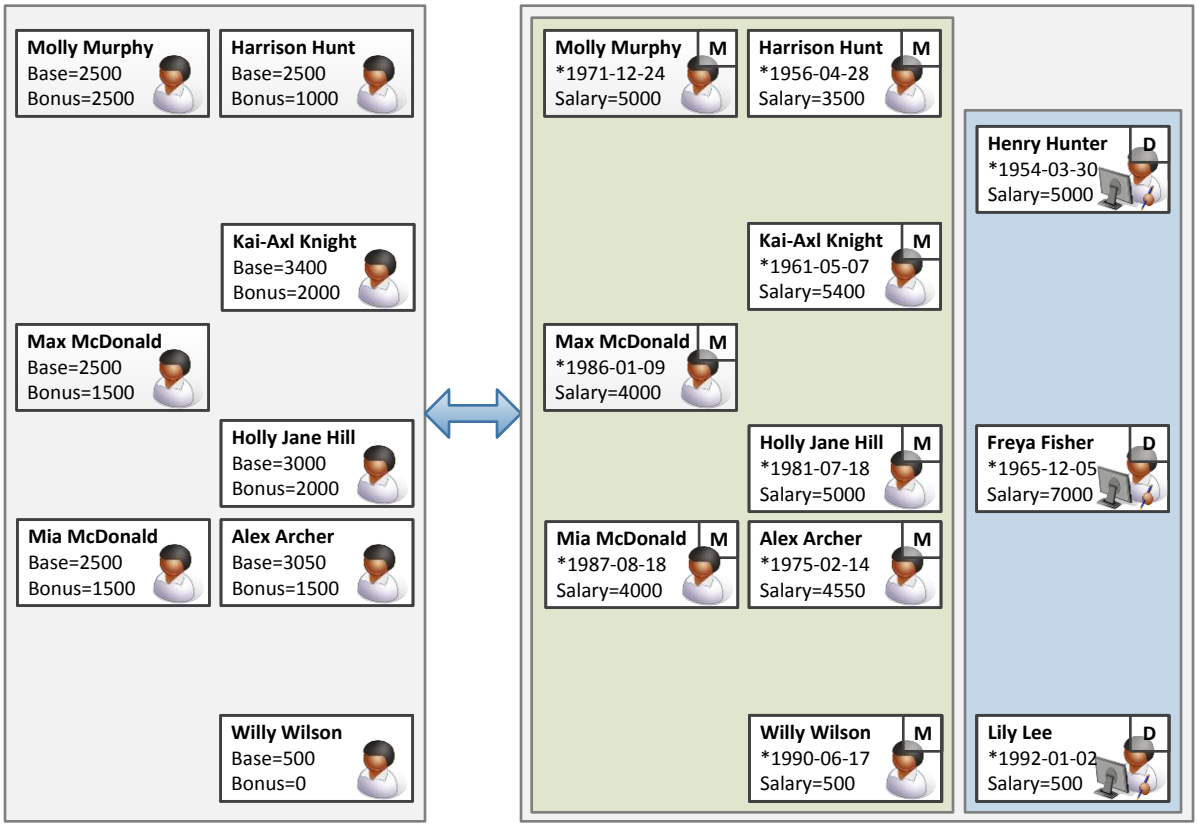


Figure 2: Concurrent model synchronization: desired result of compact example

Example 2.17 (Case 1 - Appointment of Molly Murphy)**.** Molly Murphy is employed at the marketing department by the human resource manager. After applying the model updates d_2^S and d_2^T, her data record should be available at both domains. Because no information is known about the split of her Salary (target domain) into Base and Bonus (source domain), her given Salary of 5000 will be divided into Salary / 2 = Base = Bonus. The correct division of the Salary have to be entered later by the administration of the marketing department.

Example 2.18 (Case 2 - Appointment of Harrison Hunt)**.** The appointment of Harrison Hunt at the marketing department was entered by the marketing department only. On the source domain of our

model, no possibility for entering the birth date is given, because it only reflects detailed information about the Base and Bonus salary. Nevertheless, his birth date is known, so that a possibility of manually adding the birth date during the propagation of the changes from source to target domain should be given, so that his data records will be complete on both domains of the model.

Example 2.19 (Case 3 - Resignation of Paul Page)**.** The data record of Paul Page should be deleted in the resulting model, because he resigned his work at the marketing department.

Example 2.20 (Case 4 - Transfer of Henry Hunter)**.** Henry Hunter is transferred from the marketing department to the development department. Consequently, he should be deleted from the marketing department on both domains, but added to the development department in the target domain.

Example 2.21 (Case 5 - Resignation of Lucy Lewis)**.** The resignation of Lucy Lewis is entered on the source domain. The resulting model should propagate this deletion also to the target model.

Example 2.22 (Case 6 - Renaming of K.-A. Knight)**.** The resulting model should contain the corrected first name *Kai-Axl* of employee K.-A. Knight in both domains.

Example 2.23 (Case 7 - Appointment of Max McDonald)**.** Max McDonald gets employed by the marketing department. Even though this change is entered in both domains, this modifications involves only one person, therefore Max McDonald should be added only one time to the system in the source and in the target domain and the connection between his Salary in the target domain and his Base and Bonus in the source domain should be Salary = 4000 = Base + Bonus = 2500 + 1500 where Base = 2500 and Bonus = 1500.

Example 2.24 (Case 8 - Retirement of Daniel Davis)**.** The data record of Daniel Davis shall be deleted from the source and target domain in the resulting model because Daniel Davis has retired.

Example 2.25 (Case 9 - Transfer of Freya Fisher)**.** Freya Fisher was formerly employed at the marketing department and has changed to the development department. This model update should result in the deletion of Freya Fisher's data record in the source domain, whereas in the target domain, she will be part of the development department.

Example 2.26 (Case 10 - Renaming of Holly J. Hill)**.** The resulting model should reflect the change of the abbreviated second name *J.* to *Jane* in Holly Jane Hill's data set in both domains.

Example 2.27 (Case 11 - Renaming of Mia Miller)**.** Mia Miller has married and therefore her last name changed from *Miller* to *McDonald*. This model update is entered in both domains, but in the source domain, a typing error occurred. Consequently, in the resulting model the correct last name *McDonald* should be obtained in both domains, whereas the typing error should be rejected. The decision of choosing the correct last name has to be done manually.

Example 2.28 (Case 12 - Changing the salary and bonus of Alex Archer)**.** The marketing department increases the Bonus of Alex Archer by 500 due to a great performance in his job, which was entered at the source domain. At the same time, his Salary is increased by the company in order to compensate the inflationary adjustment, which was entered in the target domain. The raise of Alex Archers Salary and Bonus take place because of different reasons, thus both types of increase shall be paid to Alex Archer. The desired new values illustrated in Fig. 2 for the Base is 3050, which contains the compensation of the inflationary adjustment. The new Bonus shall be 1500 including the increase of the Bonus by 500. The resulting Salary in the target domain should be 4550. In order to achieve this desired result, a manual intervention is necessary.

Example 2.29 (Case 13 - Resignation and changing the base salary of Jack Judge)**.** In the resulting model the data record of Jack Judge should be deleted in both domains, because he resigns from the company. The increase of the salary is a procedure, which will be executed by the administration of the marketing department in a nearly automatic way, but, in this case, it is an incorrect update. To solve the occurring conflict, a manual intervention will be necessary.

Example 2.30 (Case 14 - Resignation and changing the birth date of Tony Taylor)**.** Tony Taylor quits his work at the company, whereas a new staff member with the same name will be employed by the marketing department. This leads to a misunderstanding of the administrative staff in the target domain, who modifies the birth date of the data record of the former employee instead of adding a new one. The resulting model should reflect the correct change, i.e., the existing data record of Tony Taylor should be deleted in both domains.

Example 2.31 (Case 15 - Transfer of Lily Lee)**.** The former trainee Lily Lee is employed by the development department. The resulting model should reflect this modification in both domains, i.e., Lily Lee is deleted in the source domain, and in the target domain, she is member of the development department.

Example 2.32 (Case 16 - Switch of Willy Wilson to a permanent position)**.** Willy Wilson has been a trainee at the company and is now hired as employee at the marketing department. In the resulting model, Willy Wilson should be part of the marketing department, both in the source domain, as well as in the target domain.

3 Non-Deterministic Concurrent Synchronisation Framework

In this section, we generalise the formal framework for concurrent model synchronisation presented in [15, 12, 18]. The main task of concurrent model synchronisation is to take a given integrated model together with concurrently performed model updates in the source and target domains and to derive a consistent integrated model together with corresponding updates on both domains. Triple graph grammars (TGGs) are a suitable formal approach for defining a language of consistently integrated models [25, 4]. They have been applied successfully for (concurrent) model synchronization [7, 15, 12, 18] using the generated operations for bidirectional model transformations [26, 13]. In the framework of TGGs, an integrated model is represented by a triple graph G consisting of graphs G^S, G^C, and G^T, called source, correspondence, and target graphs, together with two mappings (graph morphisms) $s_G : G^C \to G^S$ and $t_G : G^C \to G^T$ for specifying the correspondence links. Attribute values of nodes and edges are defined as links to actual values according to [5]. The two mappings in G specify a *correspondence* $r : G^S \leftrightarrow G^T$, which relates elements of G^S with corresponding elements of G^T and vice versa.

Triple graphs are related by triple graph morphisms $m : G \to H$ [25, 4] consisting of three graph morphisms that preserve the associated correspondences (i.e., the diagrams on the right commute). Triple graphs are typed over a triple type graph TG by a triple graph morphism $type_G : G \to TG$, such that TG plays the role of a meta-model. Morphisms between typed triple graphs preserve the typing. For a triple type graph $TG = (TG^S \leftarrow TG^C \to TG^T)$, we use $VL(TG)$, $VL(TG^S)$, and $VL(TG^T)$ to denote the classes (visual languages) of all graphs typed over TG, TG^S, and TG^T, respectively.

$$\begin{array}{ccccc} G = (G^S & \xleftarrow{s_G} & G^C & \xrightarrow{t_G} & G^T) \\ {\scriptstyle m}\downarrow \quad {\scriptstyle m^S}\downarrow & & {\scriptstyle m^C}\downarrow & & {\scriptstyle m^T}\downarrow \\ H = (H^S & \xleftarrow[s_H]{} & H^C & \xrightarrow[t_H]{} & H^T) \end{array}$$

Example 3.1 (Triple Type Graph)**.** The triple type graph is illustrated in Fig. 3 in compact notation and in Fig. 4 in E-Graph notation. E-Graphs are the underlying formal representation of attributed graphs.

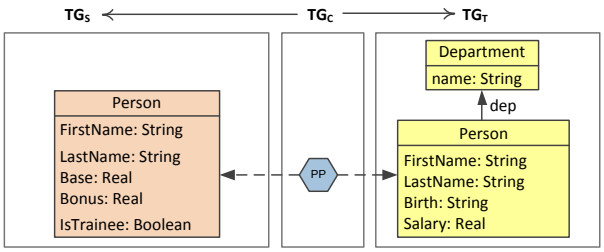


Figure 3: Triple Type Graph: Compact Notation

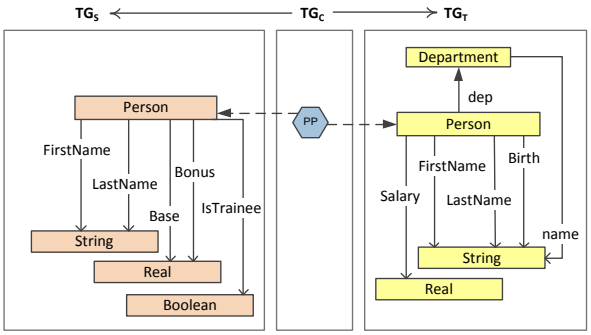


Figure 4: Triple Type Graph: E-Graph Notation

They define attributes as links from attributed elements (nodes and edges) to the actual values (c.f. [5]). According to the triple type graph, models in the source domain contain persons with their first and last name, their detailed salary information divided into base and bonus salary and their state of employment, i.e., it is given, if the person is a trainee or a permanent employee. Models in the target domain contain the first name, last name and birth date of the person and the full salary. The salary is given by an attribute without any detailed information about base and bonus salary. The membership to a department is provided by a direct link to the department. Trainees have no link to any department.

A triple graph grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph S and a set TR of triple rules, and generates the triple graph language of consistently integrated models $VL(TGG) \subseteq VL(TG)$ with consistent source and target languages $VL_S = \{G^S \mid (G^S \leftarrow G^C \rightarrow G^T) \in VL(TGG)\}$ and $VL_T = \{G^T \mid (G^S \leftarrow G^C \rightarrow G^T) \in VL(TGG)\}$. A model update $d : G \rightarrow G'$ is specified as a *graph modification* $d = (G \xleftarrow{i_1} I \xrightarrow{i_2} G')$ with inclusions $i_1 : I \hookrightarrow G$ and $i_2 : I \hookrightarrow G'$. Intuitively, all elements in $G \setminus I$ are deleted and all the elements in $G' \setminus I$ are added by d.

A triple rule $tr = (tr^S, tr^C, tr^T)$ is an inclusion of triple graph L (left hand side) in triple graph R (right hand side), represented by $tr : L \hookrightarrow R$. It specifies how a given consistent integrated model can be extended simultaneously on all three components yielding again a consistent integrated model. In particular, this means that triple rules are non-deleting. This is sufficient, because triple rules are not used for editing in the source and target domains. A triple rule is applied to a triple graph G by matching L to some subtriple graph of G via a match morphism $m : L \rightarrow G$. The result of this application is the triple graph H, where L is replaced by R in G. Technically, the result of the transformation is defined by a pushout diagram, as depicted on the right. This triple graph transformation (TGT) step is denoted by $G \overset{tr,m}{\Longrightarrow} H$. Moreover, triple rules can be extended by negative application conditions (NACs) for restricting their application to specific matches [13].

$$\begin{array}{ccc} L & \overset{tr}{\hookrightarrow} & R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle n} \\ G & \underset{t}{\hookrightarrow} & H \end{array}$$

Example 3.2 (Triple Graph Grammar)**.** The TGG of our scenario is given by the triple type graph shown in Fig. 3, the empty start graph and the set of triple rules illustrated in compact notation in Fig. 5. All elements (nodes, edges or attributes) that are marked with ++ (green border) are added by the triple rule. Parts marked with a rectangle containing the label NAC (red border) describe negative application conditions [13]. The trainee status of the person is specified by attribute IsTrainee (**T** (true) or **F** (false)) in the source domain and only full employees are linked to a department in the target domain.

The first triple rule Person2FirstMarketing inserts a new department called "Marketing" into the target domain, only if no department with the same name already exists. The negative application condition

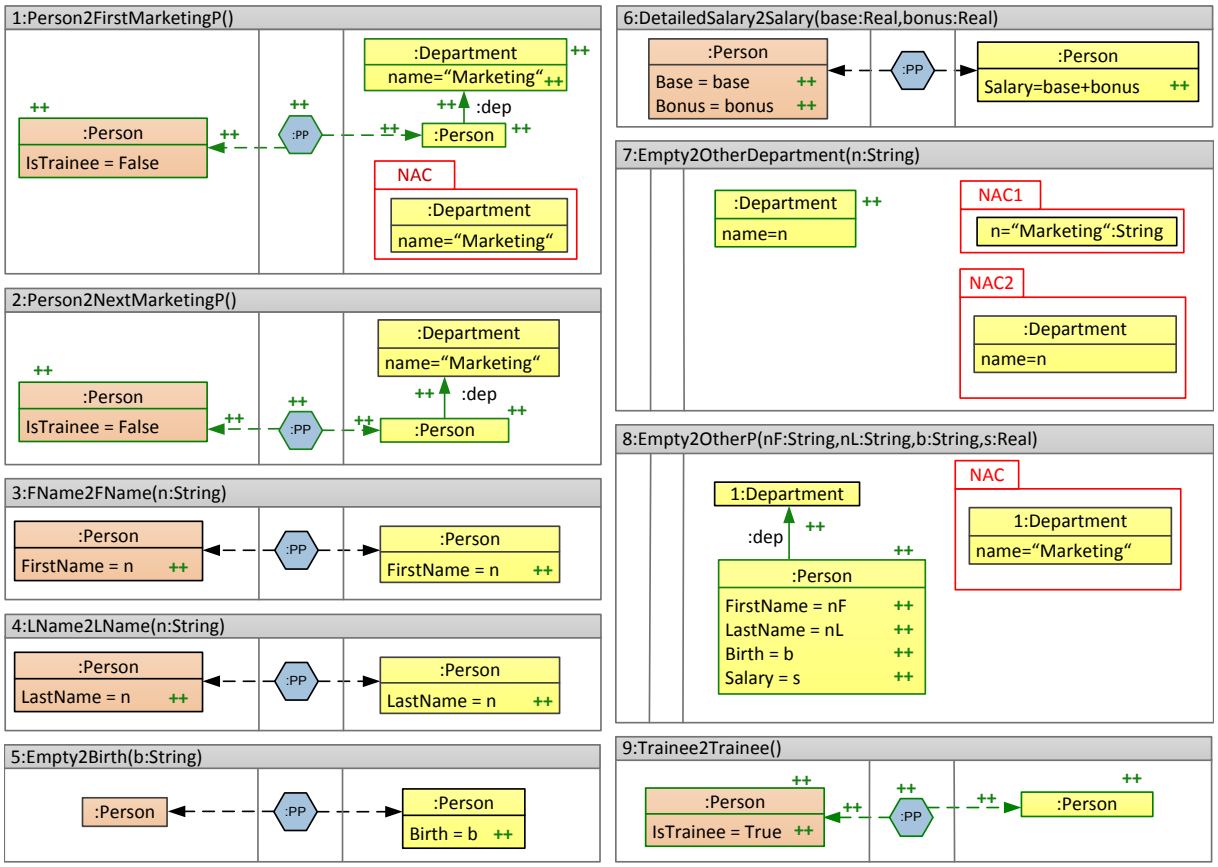


Figure 5: Triple rules

(NAC) ensures, that a department named "Marketing" will not be created twice. In addition, this rule creates a person in the target domain that is directly linked to the marketing department, which is indicated by the edge dep, in order to specify the membership of the person to the marketing department. In the source domain, the same person will be created and linked to the corresponding person in the target domain. The attribute IsTrainee is set to False, because this person is no trainee. The second triple rule Person2NextMarketingP will be applied, if in the target domain a marketing department is already created. It extends the model by a new person employed at the marketing department. For that, a person will be created in the source and target model with correspondences between them and a link of the person in the target model to the marketing department. The person is no trainee, consequently, the attribute IsTrainee is set to False. Triple rule FName2FName gets a string as input. It sets the input value as first name of a person in the source domain and of the corresponding person in the target domain. Similar to the third rule, triple rule LName2LName gets a string input and sets this as last name of a person in the source domain and of the same person in the target domain. Empty2Birth assigns the birth date given as input to the person in the target domain, because the birth date is only reflected in the target domain of the model. The salary of a person is added by the next triple rule DetailedSalary2Salary. This triple rule expects two input parameters indicating the base and bonus salary of the person, which will be both added as attributes to the person in the source model. In the target domain, the full salary of the corresponding person will be added as attribute, whose value will be the sum of the base and bonus salary.

Triple rule Empty2OtherDepartment is empty in the source and correspondence domain. It extends the target model by a new department, but only if no other department is named equally and only if the name of the new department is not "Marketing", which is ensured by both NACs. As already mentioned before, the source model only contains employees of the marketing department or trainees, whereas trainees do not belong to any department at all. Therefore, triple rule Empty2OtherP only adds a person in the target domain which is directly linked to a department which is not named "Marketing". Furthermore, all necessary attributes (first name, last name, birth date and salary) are added, which are provided by input parameters. This person is no trainee, because the person is employed at a department. This rule is empty in the source and correspondence domain, because the newly added person is no trainee and does not belong to the marketing department. With the last rule Trainee2Trainee, a trainee will be added in the source and in the target domain and connected by a correspondence. The traineeship of the person is indicated by setting the attribute IsTrainee to *True* in the source domain and by creating no link between the corresponding person and any department in the target domain.

The concurrent synchronisation problem is formalised in the left of Fig. 6. Given an integrated model $G_0 = (G_0^S \leftrightarrow G_0^T)$ and two model updates $d_1^S = (G_0^S \to G_1^S)$ and $d_1^T = (G_0^T \to G_1^T)$, we need to find source update $d_2^S = (G_1^S \to G_2^S)$ and target update $d_2^T = (G_1^T \to G_2^T)$ together with a new integrated model $G_2 = (G_2^S \leftrightarrow G_2^T)$ [12]. The solution for this problem is the concurrent synchronisation operation CSync, which is in general non-deterministic, i.e., it may require backtracking and it may yield different results for the same input.

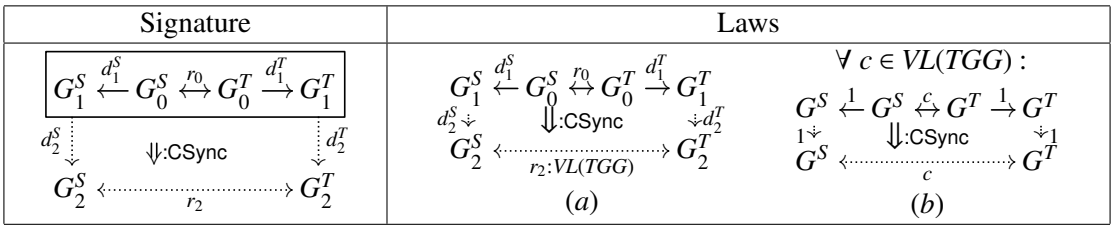


Figure 6: Signature and laws for correct concurrent synchronisation frameworks

Correctness of a concurrent synchronisation operation CSynch (right of Fig. 6) ensures that any resulting integrated model $G_2 = (G_2^S \leftrightarrow G_2^T)$ is consistent (law (*a*)), i.e., $G_2 \in VL(TGG)$. Furthermore, if the given input is initially consistent and the updates do not change anything, then one of the possible outputs of operation CSync is identical to the input itself (law (*b*)). Completeness means that *CSynch* yields at least one possible output for any input.

Definition 3.3 (Non-Deterministic Concurrent Synchronisation Problem and Framework)**.** Given a triple type graph *TG*, the *concurrent synchronisation problem* is to construct a non-deterministic operation CSync leading to the signature diagram in Fig. 6 with concurrent synchronisation operation CSync. Given a triple graph grammar $TGG = (TG, TR)$ and a concurrent synchronisation operation CSync, the non-deterministic *concurrent synchronisation framework CSynch*(TGG, CSync) is called *correct*, if laws (*a*) and (*b*) in Fig. 6 are satisfied and it is called *complete*, if operation CSync is a left total relation.

In order to review the construction of the concurrent synchronisation operation CSync, we need to recall the general notions for bidirectional model transformations based on TGGs. The corresponding *operational rules* for executing bidirectional model transformations are derived from the given TGG. In particular, TR_S and TR_F denote the sets of all source and forward rules derived from *TR* as shown below. They are used to implement source-to-target transformations. The sets of target rules TR_T and backward

rules TR_B are derived analogously and the extended construction for triple rules with negative application conditions is presented in [3].

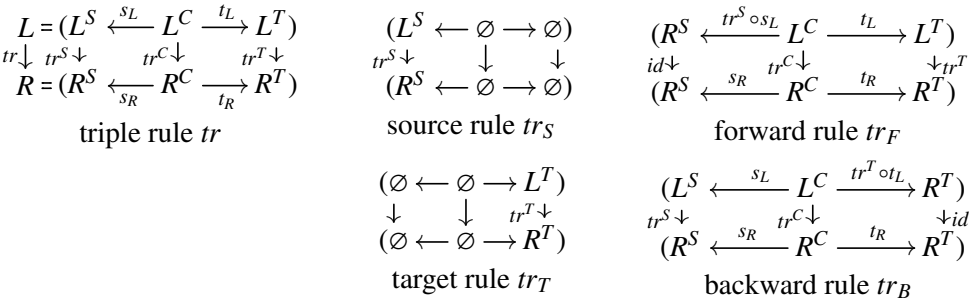


Figure 7: Operational rules of a TGG for bidirectional model transformations

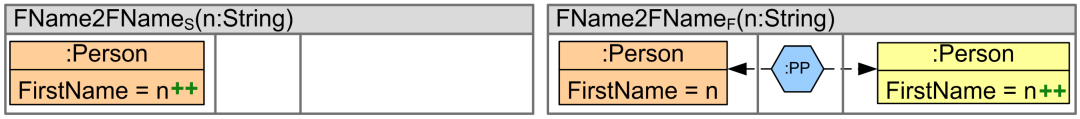


Figure 8: Derived source and forward rules

Example 3.4 (Operational Rules)**.** The rules in Fig. 8 are the derived source and forward rules of the triple rule FName2FName in Fig. 5.

As presented in [3], the derived operational rules provide the basis to define model transformations based on source consistent forward transformation sequences. Source consistency of a forward sequence $(G_0 \overset{tr_F^*}{\Longrightarrow} G_n)$ via TR_F requires that there is a corresponding source sequence $(\varnothing \overset{tr_S^*}{\Longrightarrow} G_0)$ via TR_S, such that matches of corresponding source and forward steps are compatible. The source sequence is obtained by parsing the given source model in order to guide the forward transformation. As presented in [3, 13], source and forward sequences can be constructed simultaneously and backtracking can be reduced in order to derive efficient executions of model transformations. Given a source model G^S, then a *model transformation sequence* for G^S is given by $(G^S, G_0 \overset{tr_F^*}{\Longrightarrow} G_n, G^T)$, where G^T is the resulting target model derived from the source consistent forward sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_n$ is a source-consistent forward sequence with $G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$ and $G_n = (G^S \leftarrow G^C \rightarrow G^T)$.

Model transformations based on forward rules using the control condition "source consistency" are syntactically correct and complete as shown in [3]. Correctness means that for each source model G^S that is transformed into a target model G^T there is an integrated model $G = (G^S \leftarrow G^C \rightarrow G^T)$ in the language of integrated models $VL(TGG)$ generated by the TGG. Completeness ensures that for each valid source model there is always a forward transformation sequence that transforms it into a valid target model.

In order to efficiently execute model transformations based on TGGs, the concept of translation attributes was introduced in [13] that automatically ensures source consistency during execution. We review the main technical concepts based on translation attributes used for the construction and extension of the operational rules.

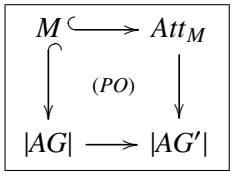
Given an attributed graph AG and a family of subsets $M \subseteq AG$ for nodes and edges, we call AG' a graph with translation attributes over AG if it extends AG with one Boolean-valued attribute `tr_x` for each element x (node or edge) in M and one Boolean-valued attribute `tr_x_a` for each attribute associated to such an element x in M. The family M together with all these additional translation attributes is denoted

by Att_M. Note that we use the attribution concept of E-graphs as presented in [5], where attributes are possible for nodes and edges. An E-graph EG extends a directed graph $G = (V, E, (s,t : E \to V))$ by a set of attribute value nodes V_D together with sets of attribution edges E_{NA} and E_{EA} for assigning attribute values to structural graph nodes V and edges E. An attributed graph $AG = (G, D)$ is given by an E-graph G together with a data algebra D, such that the attribute values V_D are given by the disjoint union of the carrier sets of D.

Definition 3.5 (Family with Translation Attributes)**.** Given an attributed graph $AG = (G, D)$ we denote by $|AG| = (V_G^G, V_G^D, E_G^G, E_G^{NA}, E_G^{EA})$ the underlying family of sets containing all nodes and edges. Let $M \subseteq |AG|$ with $(V_M^G, V_M^D, E_M^G, E_M^{NA}, E_M^{EA})$, then a *family with translation attributes* for (AG, M) extends M by additional translation attributes and is given by $Att_M = (V_M^G, V_M^D, E_M^G, E^{NA}, E^{EA})$ with:

- $E^{NA} = E_M^{NA} \cup \{\texttt{tr\_}x \mid x \in V_M^G\} \cup \{\texttt{tr\_}x\texttt{\_}a \mid a \in E_M^{NA}, src_G^{NA}(a) = x \in V_G^G\}$,
- $E^{EA} = E_M^{EA} \cup \{\texttt{tr\_}x \mid x \in E_M^G\} \cup \{\texttt{tr\_}x\texttt{\_}a \mid a \in E_M^{EA}, src_G^{EA}(a) = x \in E_G^G\}$.

Definition 3.6 (Graph with Translation Attributes)**.** Given an attributed graph $AG = (G, D)$ and a family of subsets $M \subseteq |AG|$ with $\{\mathbf{T}, \mathbf{F}\} \subseteq V_M^D$ and let Att_M be a family with translation attributes for (G, M) according to Def. 3.5. Then, $AG' = (G', D)$ is a *graph with translation attributes* over AG, where the domains $|AG'|$ of AG' are given by the gluing via pushout of $|AG|$ and Att_M over M and the source and target functions of G' are defined as follows:



- $src_{G'}^G = src_G^G$, $trg_{G'}^G = trg_G^G$,
- $src_{G'}^X(z) = \begin{cases} src_G^X(z) & z \in E_G^X \\ x & z = \texttt{tr\_}x \text{ or } z = \texttt{tr\_}x\texttt{\_}a \end{cases}$ for $X \in \{NA, EA\}$,
- $trg_{G'}^X(z) = \begin{cases} trg_G^X(z) & z \in E_G^X \\ \mathbf{T} \text{ or } \mathbf{F} & z = \texttt{tr\_}x \text{ or } z = \texttt{tr\_}x\texttt{\_}a \end{cases}$ for $X \in \{NA, EA\}$.

Att_M^v, where $v = \mathbf{T}$ or $v = \mathbf{F}$, denotes a family with translation attributes where all attributes are set to v. Moreover, we denote by $AG \oplus Att_M$ that AG is extended by the translation attributes in Att_M, i.e. $AG \oplus Att_M = (G', D)$ for $AG' = (G', D)$ as defined above. We use the notion $AG \oplus Att_M^v$ for translation attributes with value v and we use the short notion $Att^v(AG) := AG \oplus Att_{|AG|}^v$.

As shown in [13], the construction of forward and backward translation rules allows for the efficient implementation and analysis of model transformations. In combination with the generation consistency creating rules, the sets of operational translation rules provide the basis for concurrent model synchronisation [18]. The consistency creating rules are used to partially parse a given triple model G, where partial parsing means finding a maximal consistent submodel $G_0 \subseteq G$.

Definition 3.7 (Operational Translation Rules)**.** Given a triple rule $tr = (L \to R)$ and its derived source rule $tr_S = (L_S \to R_S)$, target rule $tr_T = (L_T \to R_T)$, forward rule $tr_F = (L_F \to R_F)$ and backward rule $tr_B = (L_B \to R_B)$, the derived translation rules of tr are given by *consistency creating rule* $tr_{CC} = (L_{CC} \xleftarrow{l_{CC}} K_{CC} \xrightarrow{r_{CC}} R_{CC})$, *forward translation rule* $tr_{FT} = (L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT})$, and *backward translation rule* $tr_{BT} = (L_{BT} \xleftarrow{l_{BT}} K_{BT} \xrightarrow{r_{BT}} R_{BT})$ defined in Fig. 9 using the notation based on translation attributes. By TR_{CC}, TR_{FT}, TR_{BT} we denote the sets of all derived consistency creating, forward translation and backward translation rules, respectively.

	main components	new NAC for each $n : L \to N$ of tr
tr_{CC}	$L_{CC} \xleftarrow{l_{CC}} K_{CC} \xrightarrow{r_{CC}} R_{CC}$ $\| \quad \| \quad \|$ $(R \oplus Att^{\mathbf{T}}_{L} \oplus Att^{\mathbf{F}}_{R\setminus L})\ (R \oplus Att^{\mathbf{T}}_{L})\ (R \oplus Att^{\mathbf{T}}_{L} \oplus Att^{\mathbf{T}}_{R\setminus L})$	$N_{CC} = (L_{CC} +_L N) \oplus Att^{\mathbf{T}}_{N\setminus L}$
tr_{FT}	$L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT}$ $\| \quad \| \quad \|$ $(L_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{F}}_{R_S\setminus L_S})\ (L_F \oplus Att^{\mathbf{T}}_{L_S})\ (R_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{T}}_{R_S\setminus L_S})$	$N_{FT} = (L_{FT} +_L N) \oplus Att^{\mathbf{T}}_{N_S\setminus L_S}$
tr_{BT}	$L_{BT} \xleftarrow{l_{BT}} K_{BT} \xrightarrow{r_{BT}} R_{BT}$ $\| \quad \| \quad \|$ $(L_B \oplus Att^{\mathbf{T}}_{L_T} \oplus Att^{\mathbf{F}}_{R_T\setminus L_T})\ (L_B \oplus Att^{\mathbf{T}}_{L_T})\ (R_B \oplus Att^{\mathbf{T}}_{L_T} \oplus Att^{\mathbf{T}}_{R_T\setminus L_T})$	$N_{BT} = (L_{BT} +_L N) \oplus Att^{\mathbf{T}}_{N_T\setminus L_T}$

Figure 9: Components of derived operational translation rules

Remark 3.8 (Construction of Operational Rules)**.** Note that in Fig. 9 $(B +_A C)$ is the union of B and C with shared A, such that for instance $(L_{FT} +_L N)$ is the union of L_{FT} and N with shared L. Also, $G \oplus Att^{\mathbf{T}}_{M}$ denotes adding to the graph G translation attributes for all the elements and attributes included in $M \subseteq G$, and moreover all these attributes are set to **T**. Similarly, $G \oplus Att^{\mathbf{F}}_{M}$ denotes adding to G all these attributes, but this time they are set to **F**.

Due to the modification of the translation attributes, the rules are *deleting*, which means that, technically, the rules cannot be denoted by inclusions $L \hookrightarrow R$. As a consequence, from a formal point of view, triple transformations are not defined as a pushout, but in terms of the classical double pushout (DPO) approach [5].

According to Def. 3.7, the consistency creating rule does not modify the structure of a triple graph, but modifies the translation attributes only. It is used for marking consistent substructures of a given triple graph, i.e., of a given integrated model. By applying all derived consistency creating rules as long as possible to a given triple G graph with all translation attributes set to "**F**", a maximal consistent triple graph that is contained in G is computed. Intuitively, for each element $x \in R$ (node, edge, or attribute) of a triple rule $tr = (L \to R)$ a separate translation attribute (`tr` or `tr_`x) is added for the consistency creating rule tr_{CC}. If an element $x \in R$ is preserved by the triple rule tr ($x \in L$), then the consistency creating rule preserves it as well and the translation attribute has value **T**. Otherwise, if $x \in R$ is created by tr ($x \in R \setminus L$), then it becomes a preserved element in the consistency creating rule tr_{CC} and the corresponding translation attribute is changed from **F** to **T**. In visual notation, this means that all plus signs are replaced by additional translation attributes whose values are changed from **F** to **T** and we denote such a modification by [**F** ⇒ **T**].

A forward translation rule tr_{FT}, introduced in [19], extends the forward rule tr_F by additional Boolean valued translation attributes, which are markers for elements in the source model and specify whether the elements have been translated already. Each forward translation rule tr_{FT} turns the markers of the source elements that are translated by this rule from **F** to **T** (i.e., the elements that are created by tr_S). This way, we can ensure that each element in the source graph is not translated twice, but exactly once. The backward translation rules are dual to the case of forward translation rules and used for the translation of target models into their corresponding source models. Thus, they do only modify translation attributes

on the target component.

Example 3.9 (Derived Sets of Operational Translation Rules)**.** Figure 16 concerns the original triple rule Trainee2Trainee and shows the derived consistency creating rule Trainee2Trainee$_{CC}$, the derived backard translation rule Trainee2Trainee$_{BT}$ (left part of the figure) and the extended versions of both rules containing additional filter NACs (right part of the figure).

Remark 3.10 (Interdependencies between Operational Rules)**.** The consistency creating rules (TR_{CC}) are used in Sec. 3 for marking the already consistent parts of a given integrated model in the second sub-phase of the synchronization. The forward and backward translation rules are used for the third sub-phase. This third sub-phase can be interpreted as a completion of the computed sequence of the second sub-phase. Since we consider non-deterministic sets of operational rules in general, this completion may fail leading to backtracking some steps of the first sequence.

We recall the definition of model transformations based on forward translation rules according to [13]. A model transformation sequence starts with a triple graph that consists only of the given source graph, i.e., the target and the connection graphs are the empty graphs. At the beginning, the source graph is completely marked with **F**-valued translation attributes indicating that no element from the graph has been translated yet. Then, we apply a sequence of forward translation transformation steps leading to a graph whose source part is completely marked with **T** meaning that all the elements from the given source graph have been translated. These transformation sequences are called *complete forward translation sequences*.

Definition 3.11 (Complete Forward Translation Sequence)**.** A forward translation sequence $G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n$ with almost injective matches is called *complete* if G_n is *completely translated*, i.e., all translation attributes of G_n are set to true ("**T**").

A *model transformation based on forward translation rules* transforms models from the source domain into models of the target domain by executing complete forward translation sequences. Given a concrete source model, then the resulting target model of the model transformation is obtained by restricting the resulting triple graph of the forward translation sequence to the target component. We have shown in [13] that model transformation sequences based on forward rules and those based on forward translation rules, respectively, are equivalent. This ensures that the derived model transformation relations are the same.

Definition 3.12 (Model Transformation Based on Forward Translation Rules)**.** A *model transformation sequence* $(G^S, G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n, G^T)$ based on forward translation rules TR_{FT} consists of a source graph G^S, a target graph G^T, and a complete TGT-sequence $G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n$ typed over $TG' = TG \oplus Att^{\mathbf{F}}_{|TG^S|} \oplus Att^{\mathbf{T}}_{|TG^S|}$ based on TR_{FT} with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ and $G'_n = (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.
A *model transformation* $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ based on TR_{FT} is defined by all model transformation sequences as above with $G^S \in VL(TG^S)$ and $G^T \in VL(TG^T)$. All the corresponding pairs (G^S, G^T) define the *model transformation relation* $MTR_{FT} \subseteq VL(TG^S) \times VL(TG^T)$ based on TR_{FT}. The model transformation is *terminating* if there are no infinite TGT-sequences via TR_{FT} starting with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ for some source graph $G^S \in VL(TG^S)$.

Consistency creating sequences as defined in Def. 3.13 below, are used for computing a maximal consistent part of a given triple graph, which is used for the auxiliary operation Del in Sec. 3. A consistency creating sequence starts at a triple graph $G'_0 = Att^{\mathbf{F}}(G)$, i.e., at a triple graph where all elements are marked with **F**. Each application of a consistency creating rule modifies some translation attributes of an

intermediate triple graph G'_i from **F** to **T** and preserves the structural part G contained in G'_i. Therefore, the resulting triple graph G'_n extends G with translation attributes only, i.e., some are set to **T** and the remaining ones to **F**.

Definition 3.13 (Consistency Creating Sequence)**.** Given a triple graph grammar $TGG = (TG, \emptyset, TR)$, a triple graph G typed over TG and let TR_{CC} be the set of consistency creating rules of TR. A *consistency creating sequence* $s = (G, G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_n, G_n)$ is given by a TGT sequence $G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_n$ via TR_{CC} with $G'_0 = Att^{\mathbf{F}}(G)$ and $G'_n = G \oplus Att^{\mathbf{T}}_{G_n} \oplus Att^{\mathbf{F}}_{G \setminus G_n}$, where G_n is the subgraph of G derived from $G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_n$ by restricting G'_n to all **T**-marked elements. Consistency creating sequence s is called *terminated*, if there is no rule in TR_{CC} which is applicable to the result graph G'_n. In this case, the triple graph G'_n is called a maximal consistency marking of G. A triple graph G' is called *completely* **T***-marked*, if $G' = Att^{\mathbf{T}}(G)$ for a given triple graph G, i.e., all translation attributes in G' are "**T**".

Remark 3.14 (Termination)**.** It is quite easy to show that, unless the given TGG includes a trivial identical rule $L \hookrightarrow L$, every consistency creating sequence terminates. The reason is that the application of each rule switches some translation predicates from **F** to **T**. Since the number of these predicates in a given triple graph is finite, only a finite number of rule applications is possible.

The case of forward and backward translation sequences is different. In particular, if a triple rule $tr = L \hookrightarrow R$ is *source identic*, meaning that it does not change the source part, i.e., $L^S = R^S$ or equivalently $tr^s = id$, its associated forward translation rule will not switch any translation predicate from **F** to **T**. This implies that this rule could be applied infinitely many often in a forward translation sequence. Something similar happens with backward translation rules.

In this sense, according to whether rules modify or not the source or target part of a rule, we classify rules as shown below. In particular, this notation is used in the following section. Let TR be a set of triple rules. We distinguish the following subsets.

- The set of *source creating* rules $TR^{+s} = \{tr \in TR \mid tr^S \neq id\}$,
- The set of *source identic* rules $TR^{1s} = \{tr \in TR \mid tr^S = id\}$,
- The set of *target creating* rules $TR^{+t} = \{tr \in TR \mid tr^T \neq id\}$,
- The set of *target identic* rules $TR^{1t} = \{tr \in TR \mid tr^T = id\}$, and
- The set of *identic* rules $TR^1 = \{tr \in TR \mid tr = id\}$.

In order to ensure termination for forward translation sequences, if the given TGG includes source identic triple rules, we propose a general strategy based on an automated analysis using the tool AGG. The main idea is the following. If we can show that none of the remaining triple rules depends on the source identic triple rules, we can actually omit the source identic ones. The reason is that for each forward transformation sequence, we can shift the steps along source identic rules to the end and obtain an equivalent sequence. Since all steps along source identic triple rules do not change the marking of the source model, we further derive that these steps can be removed yielding still a complete forward translation sequence.

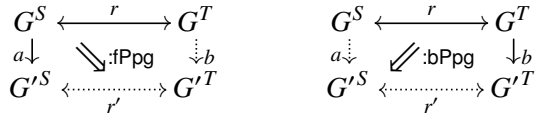


Figure 10: Propagation operations

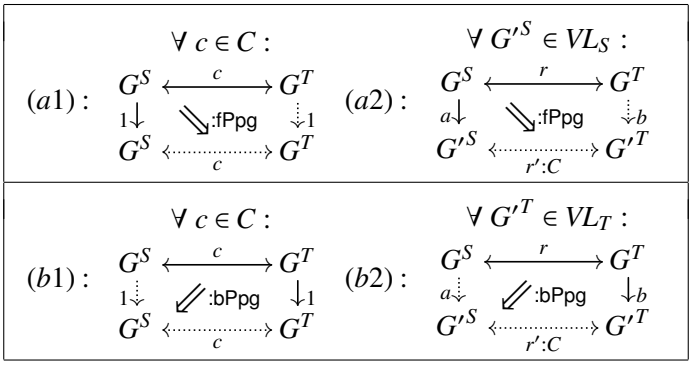


Figure 11: Laws for correct and complete propagation operations

Given a TGG, *the unidirectional synchronization problem* is to provide suitable non-deterministic forward and backward operations fPpg and bPpg that propagate updates on one model (G^S or G^T) to the other model. More precisely, given an integrated model (a correspondence relation) $G^S \leftrightarrow G^T$ and an update $a : G^S \to G'^S$, the operation fPpg propagates the update a to G^T returning as results an update $b : G^T \to G'^T$ and a correspondence relation $G'^S \leftrightarrow G'^T$. Similarly, bPpg is the dual operation that propagates updates on target models to updates on source models. The effect of these operations is depicted schematically in the diagrams on Fig. 10, where we use solid lines for the inputs and dashed lines for the outputs. The propagation operations are *correct*, if they additionally preserve consistency as specified by laws $(a1)-(b2)$ in Fig. 11. Law $(a2)$ means that fPpg always produces consistent correspondences from consistent updated source models G'^S. Law $(a1)$ means that if the given update is the identity and the given correspondence is consistent, then fPpg changes nothing. Laws $(b1)$ and $(b2)$ are the dual versions concerning bPpg. Moreover, the sets VL_S and VL_T specify the *consistent source and target models*, which are given by the source and target components of the integrated models in $C = VL(TGG)$.

Definition 3.15 (Unidirectional Synchronization Operations)**.** Given a triple graph grammar *TGG*, the *unidirectional forward synchronization problem* is to construct a non-deterministic propagation operation $\mathsf{fPpg} : R \otimes \Delta_S \to R \times \Delta_T$ leading to the left diagram in Fig. 10, where $R \otimes \Delta_S = \{(r,a) \in R \times \Delta_S \mid r\colon G^S \leftrightarrow G^T, a\colon G^S \to G'^S\}$, i.e., a and r coincide on G^S. The pair $(r,a) \in R \otimes \Delta_S$ is called premise and $(r',b) \in R \times \Delta_T$ is called solution of the forward synchronization problem, written $\mathsf{fPpg}(r,a) = (r',b)$. The *unidirectional backward synchronization problem* is to construct a non-deterministic operation bPpg leading to the right diagram in Fig. 10. Operation fPpg is called *correct* with respect to C, if axioms $(a1)$ and $(a2)$ in Fig. 11 are satisfied and, symmetrically, bPpg is called *correct* with respect to C, if axioms $(b1)$ and $(b2)$ are satisfied and called *complete*, if $(\mathsf{fPpg}, \mathsf{bPpg})$ are left-total relations.

The execution of operation fPpg in the case of non-deterministic sets of operational translation rules $(TR_{CC}, TR_{FT}, TR_{BT})$ is mainly performed as defined in [18], but may include backtracking. We describe the execution in detail in Rem. 3.16 below.

Remark 3.16 (Execution of Non-Deterministic Forward Propagation)**.** In the first step of operation fPpg in Fig. 13, the dangling correspondences are removed by the forward alignment operation fAln leading to a new integrated model $D^S \leftrightarrow G^T$. This first step is performed via a pullback construction according to Fig. 12, which visualises the details of the construction of the auxilary operations fAln, Del and fAdd. The second step via operation Del marks all elements of the integrated model that are still consistent prepares for the deletion of the remaining inconsistent elements. This leads to a corresponding triple

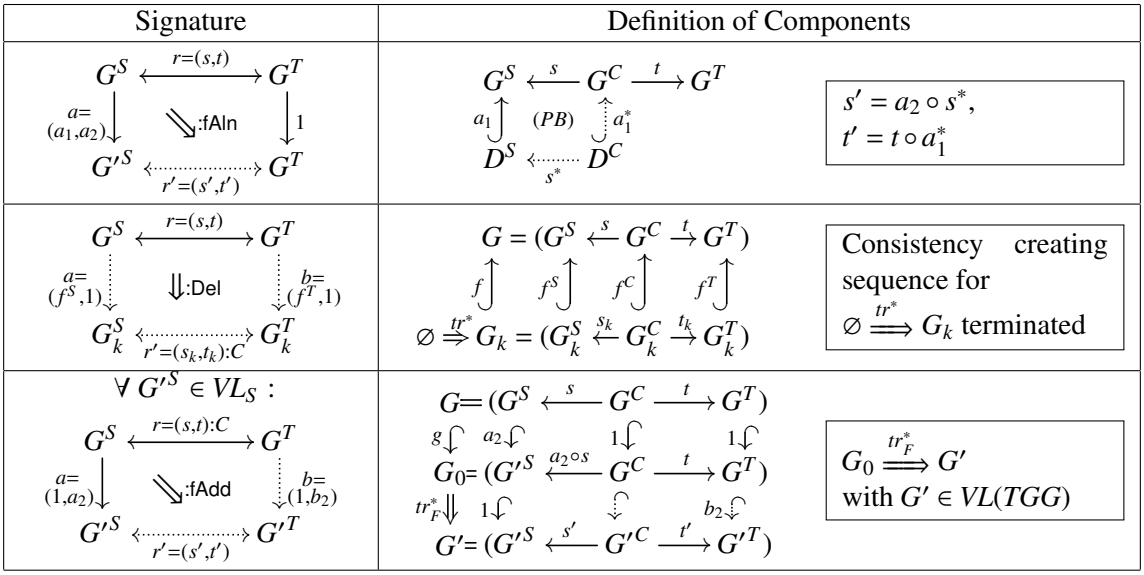


Figure 12: Auxiliary operations fAln, Del and fAdd

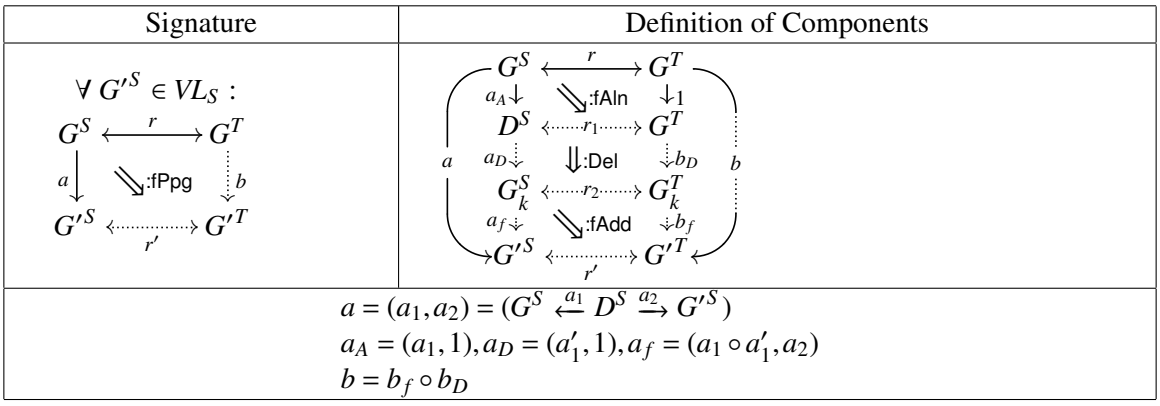


Figure 13: Synchronization operation fPpg - formal definition

sequence ($\varnothing \overset{tr^*}{\Longrightarrow} G_k$) with consistent integrated model G_k. The construction of this sequence is performed by taking the current integrated model $D^S \leftrightarrow G^T$, marking all elements with translation markers $tr = \mathbf{F}$ and applying the rules in TR_{CC} as long as possible. Since we do not require that the set TR_{CC} is deterministic, the derived transformation sequence is in general not unique. Due to the composition and decomposition result for triple graph grammars [4, 15, 18], there is a corresponding forward sequence $G_0 \overset{tr_F^*}{\Longrightarrow} G_k$ with $G_0 = (G_k^S \leftarrow \varnothing \rightarrow \varnothing)$. This sequence is extended in the third step via operation fAdd to $G'_0 \overset{tr_{F,1}^*}{\Longrightarrow} G'_k \overset{tr_{F,2}^*}{\Longrightarrow} G'_n$, where $G'_0 = (G'^S \leftarrow G_k^C \rightarrow G_k^T)$. Due to non-determinism of the operational rules, this sequence is not always source consistent. In that case, it does not specify a valid forward model transformation sequence. Therefore, we have to apply backtracking. This backtracking is successful, because the completeness result for model transformations based on forward rules ensures that there is a source consistent forward sequence $s_2 = (G'_0 \overset{tr_{F,3}^*}{\Longrightarrow} G'_n)$. Note that this means that we may also have to

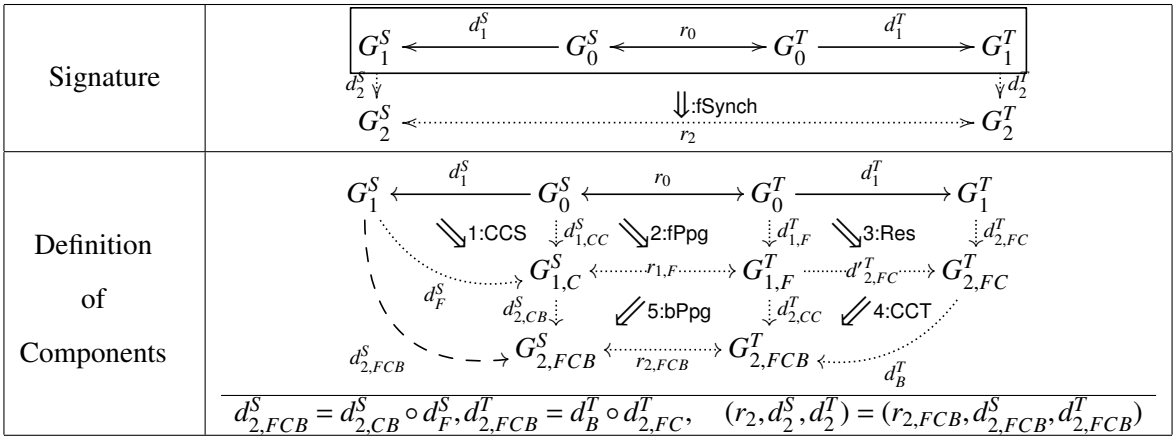


Figure 14: Concurrent model synchronization with conflict resolution (forward case: fSynch)

backtrack steps of sequence $s_1 = (G'_0 \overset{tr^*_{F,1}}{\Longrightarrow} G'_k)$ obtained from step 2 via operation Del. This means that we derive from s_1 a separation into two sub-sequences $s_{1a} = (G'_0 \overset{tr^*_{F,1a}}{\Longrightarrow} G'_i)$ and $s_{1b} = (\overset{tr^*_{F,1b}}{\Longrightarrow} G'_k)$, where s_{1a} is the part that is preserved in s_2 and s_{1b} is the part that was reverted due to possible backtracking. From these sequences, we directly derive the required modifications $a_f \colon G^S_k \to G'^S$ and $b_f \colon G^T_k \to G'^T$.

Note that in this paper, we require that the intermediate triple sequence $(\varnothing \overset{tr^*}{\Longrightarrow} G_k)$ in step 2 (operation Del) has terminated but not that the corresponding triple sequence is maximal. The reason is that the addition of filter NACs in Sec. 4 may reduce the possible sequences, such that the corresponding triple sequence without filter NACs may not ensure the maximality condition required in [12]. However, if no filter NACs are present, the marking sequence via TR_{CC} ensures the maximality condition for the triple sequence, i.e., the sequence cannot be extended to $(\varnothing \overset{tr^*}{\Longrightarrow} G_k \overset{tr_i}{\Longrightarrow} G_{k+1})$ with $G_{k+1} \subseteq (D^S \leftrightarrow G^T)$.

According to Fig. 13 and Rem. 3.16, forward propagation operation fPpg is executed in three steps via auxilliary operations fAln, Del, and fAdd based on forward translation rules. The backward propagation operation bPpg is defined symmetrically based on backward translation rules.

The formal approach to concurrent model synchronisation based on TGGs [12] is performed in five steps (see Fig. 14 and Fig. 15) and is initiated in one domain. We describe the forward case (operation fSync), i.e., the synchronisation is initiated in the source domain. The symmetric backward case (bSync, Fig. 15) works analogously by switching the roles of source and target domains and exchanging fPpg with bPpg, and CCS (consistency creating on source domain) with CCT (consistency creating on target domain). The concurrent synchronisation operation CSync = (fSync ∪ bSync) is defined by the union of both cases. Additional details for the applied auxiliary operations fPpg and bPpg are described in Rem. 3.16.

Concept 1 (Execution of non-deterministic synchronisation framework)**.** *In contrast to previous work [12], we do not require deterministic TGGs, such that all five steps may yield several results and steps 1,2,4, and 5 may require backtracking. Some examples are discussed in detail in Ex. 5.8-5.14 in the following section. The first step (*1:CCS*) is executed via consistency creating operation* CCS *on the source domain and computes the maximal sub-model $G^S_{1,C} \in VL_S$ of the given model G^S_1 that is consistent with respect to the language VL_S (language of consistent source models). We obtain source update $d_{1,CC} \colon G^S_0 \to G^S_{1,C}$. In general, consistency creating operations* CCS *(source domain, step 1) and* CCT *(target domain, step 4) remove structures that cannot be translated, i.e., those that*

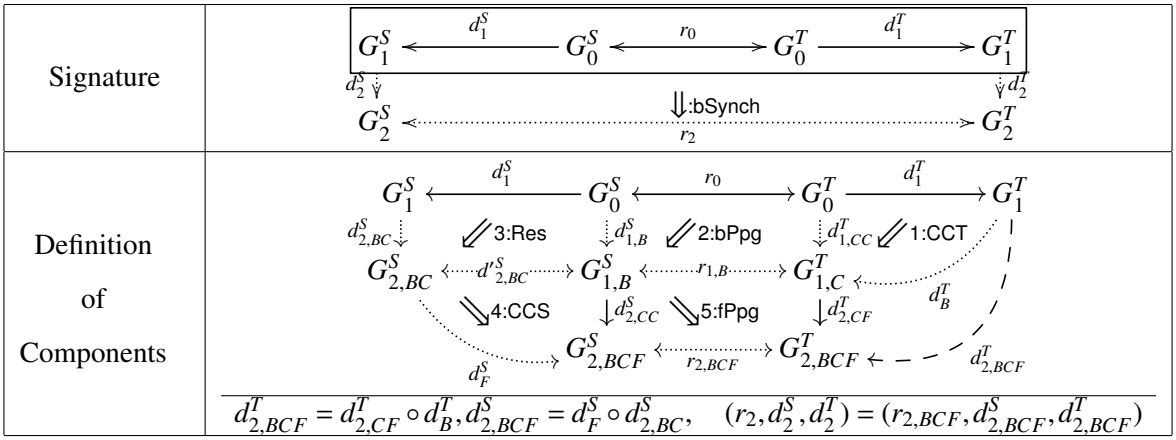


Figure 15: Concurrent model synchronization with conflict resolution (backward case: bSynch)

cannot appear in any consistent integrated model. In step 2, we apply forward propagation operation fPpg *to propagate the changes to the target domain and we obtain target update* $d^T_{1,F}$ *and integrated model* $G_{1,F} = (G^S_{1,C} \leftrightarrow G^T_{1,F})$*. In step 3, we apply conflict resolution operation* Res *[12] in order to merge the two updates on the target domain: the propagated update* $d^T_{1,F}$ *and the given update* d^T_1*. This leads to a new target update* $d'^T_{2,FC}\colon G^T_{1,F} \to G^T_{2,FC}$*. We apply consistency creating operation* CCT *to obtain the maximal consistent sub-model of* $G^T_{2,FC}$ *and derive the target updates* $d^T_B\colon G^T_{2,FC} \to G^T_{2,FCB}$ *and* $d^T_{2,CC}\colon G^T_{1,F} \to G^T_{2,FCB}$*. Finally, we propagate update* $d^T_{2,CC}$ *from the target to the source domain via backward propagation operation* bPpg *leading to source update* $d^S_{2,CB}\colon G^S_{1,C} \to G^S_{2,FCB}$ *and integrated model* $G_{2,FCB} = (G^S_{2,FCB} \leftrightarrow G^T_{2,FCB})$*.*

In order to simplify the synchronization, modellers on both domains can update their models before hand using the consistency creating operations for marking the structures to modify, such that the given source and target models G^S_1 and G^T_1 are not modified by by the synchronization operation.

Our first main result in Thm. 1 below shows that the non-deterministic concurrent synchronisation framework is correct and complete. This means that the framework yields a consistent output for any valid input. Note that termination of the synchronisation is ensured, if each operation translation rule changes at least one translation attribute [18].

Theorem 1 (Correctness and Completeness of Non-Deterministic Concurrent Synchronization Framework)**.** Given a triple graph grammar *TGG*, the derived non-deterministic concurrent synchronisation framework *CSynch*(*TGG*, CSynch) is correct and complete.

Proof. By Thm. 1 in [12] we know that the concurrent synchronisation framework is correct and complete for deterministic sets $TR_{CC}, TR_{FT}, TR_{BT}$ of operational rules. We have to show that the extended operations fPpg and bPpg based on non-deterministic sets $TR_{CC}, TR_{FT}, TR_{BT}$ are correct and complete as well. Since operations fPpg and bPpg are defined symmetrically, it is sufficient to show correctness and completeness of operation fPpg. By Rem. 3.16, we described the execution of operation fPpg and defined the difference due to backtracking. Using the correctness and completeness result for model transformations based on forward rules (Thm. 1 in [3]), we know that there is a source consistent forward sequence yielding a consistent integrated model $G' \in VL$, if $G'^S \in VL_S$. According to the preconditions in laws (*a*) and (*b*) in Fig. 11 we have that $G'^S \in VL_S$ holds and thus, operation fPpg is correct and complete. By symmetry of the definitions, we derive that operation bPpg is correct and complete.

We now consider the remaining steps in Fig. 14. Consistency checking operations (CCS, CCT) compute maximal sub models. These operations may also require backtracking, because the sets of operational rules are not necessarily deterministic. Their execution is performed by constructing a corresponding model transformation sequence. Hence, as in the deterministic case, these operations lead to the consistent models $G^S_{1,C} \in VL_S$ and $G^T_{2,FCB} \in VL_T$ as required for operations fPpg and bPpg. Moreover, if the given models are consistent already, then there is a corresponding model transformation sequence that translates this model due to the completeness result of model transformations based on TGGs (Thm. 1 in [3]). This ensures law (*b*) in Fig. 6. Finally, operation Res does not have to ensure special properties. All together, operation fSynch always yields a consistent integrated model G_2 (correctness law (a) in Fig. 6), does not change anything for consistent inputs with identical updates (correctness law (b) in Fig. 6). and it provides an output for any input (completeness).

By symmetry of the definitions, we can also conclude that operation bSynch is correct an complete (*Fig.* 15), such that operation CSynch = bSynch ∪ bSynch is correct and complete. □

4 Efficiency Improvement

The general non-deterministic concurrent model synchronisation framework in Sec. 3 may require backtracking depending on the given TGG. In order to reduce and possibly eliminate backtracking, we show in this section how to eliminate conflicts between the operational rules of a TGG using the concept of filter NACs [13], such that the correctness and completeness results are preserved.

Intuitively, a filter NAC specifies a context of a translation rule, which will always lead to an incomplete translation [13]. This means that applying the rule in any context containing the NAC pattern will require to backtrack this step. Thus, the filter NAC avoids the rule to be applied in those cases. The following example describes how the operational rules of our example scenario are extended with filter NACs.

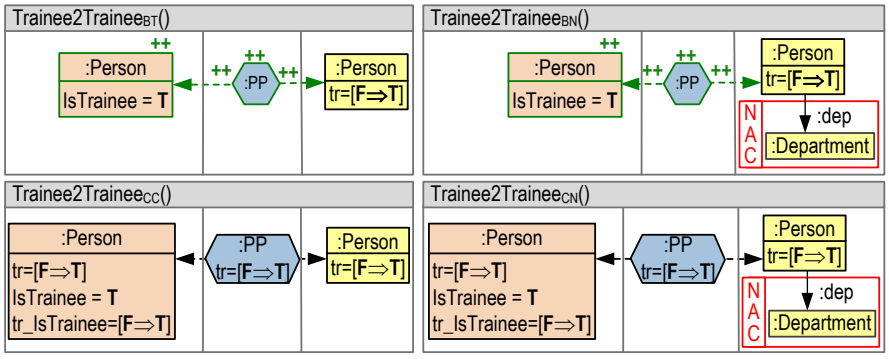


Figure 16: Some operational rules and extended operational rules

Example 4.1 (Reduction and Elimination of Backtracking)**.** The derived set of backward translation rules for the example TGG is not deterministic and backtracking is necessary for the backward propagation operation bPpg. Consider a given target model that contains a person of the marketing department that is not yet propagated (translated). Rule Trainee2Trainee$_{BT}$ (top left in Fig. 16) is applicable. However, the rule should not be applied, because the person belongs to a concrete department implying that the person is not a trainee. An application of the rule would lead to a translation sequence where the adjacent edge of type :dep remains untranslated. If instead one of the two rules {Person2FirstMarketing$_{BT}$, Person2NextMarketing$_{BT}$} that are derived from the triple rules in Fig. 5

would be applied, then the problematic edge would be translated in the same step. Thus, the backward translation requires backtracking.

The solution is to introduce a filter NAC for the rule Trainee2Trainee$_{BT}$ that avoids an application to nodes of type Person that belong to a marketing department. Using the automated generation technique for filter NACs in [13], we derive the backward translation rule Trainee2Trainee$_{BN}$ (top right in Fig. 16). In fact, the new set of backward translation rules does not require backtracking, which can be checked with the automated analysis using the tool AGG as described in [13].

However, the introduction of filter NACs causes a new problem for model synchronisation. The reason is that the consistency creating rules TR_{CC} used for marking the already consistent elements in the given integrated model are no longer compatible with the modified backward translation rules. In fact, there are cases, in which the marking via the consistency creating rules TR_{CC} mark too many elements to be consistent as shown in Ex. 4.2. In order to solve this problem, we also extend the set of consistency creating rules with additional NACs.

Example 4.2 (Extension of Consistency Creating Rules)**.** The consistency creating rule Trainee2Trainee$_{CC}$ (bottom left of Fig. 16) is used for marking consistent occurrences of a trainee in both domains. However, the rule as it is, might mark too many fragments to be consistent to complete the backward propagation during synchronisation. Consider the last case in our running example illustrated in Fig. 1 and described in Ex. 2.16. Trainee Willy Wilson becomes a full employee. Therefore, he will be assigned to a department. The initial target update would be an addition of an edge of type : dep. This would mean that the consistency creating rules will still mark every element of the integrated model G to be consistent except the new edge. However, the backward translation can not continue at this point, because the additional edge cannot be translated separately by any rule. Practically, the edge requires that the node for the former trainee is translated to a full employee (IsTrainee = **F**). Therefore, the marking step that marked the given value IsTrainee = **T** on the source domain to be consistent needs to be revoked independent of the additional context. By introducing a corresponding NAC, we derive the consistency creating rule Trainee2Trainee$_{CN}$ (bottom right of Fig. 16) and avoid this situation.

Filter NACs improve the efficiency of the execution of model transformations by cutting off possible backtracking paths [13]. They are based on the following notion of misleading graphs, which can be seen as model fragments that are responsible for the backtracking of a model transformation.

Definition 4.3 (Translatable and Misleading Graphs)**.** A triple graph with translation attributes G is *translatable* if there is a transformation sequence $G \overset{tr^*_{FT}}{\Longrightarrow} H$ via forward translation rules such that H is completely translated, i.e., all translation attributes have the value **T**. A triple graph with translation attributes G is *misleading*, if there is no triple graph that can be extended to a triple graph with translation attributes $G' \supseteq G$, such that G' is translatable.

Definition 4.4 (Filter NAC)**.** A filter NAC n for a forward translation rule $tr_{FT} : L_{FT} \leftarrow K_{FT} \rightarrow R_{FT}$ is given by a morphism $n : L_{FT} \rightarrow N$, such that there is a TGT step $N \overset{tr_{FT},n}{\Longrightarrow} M$ with M being misleading. The extension of tr_{FT} by some set of filter NACs is called forward translation rule tr_{FN} with filter NACs.

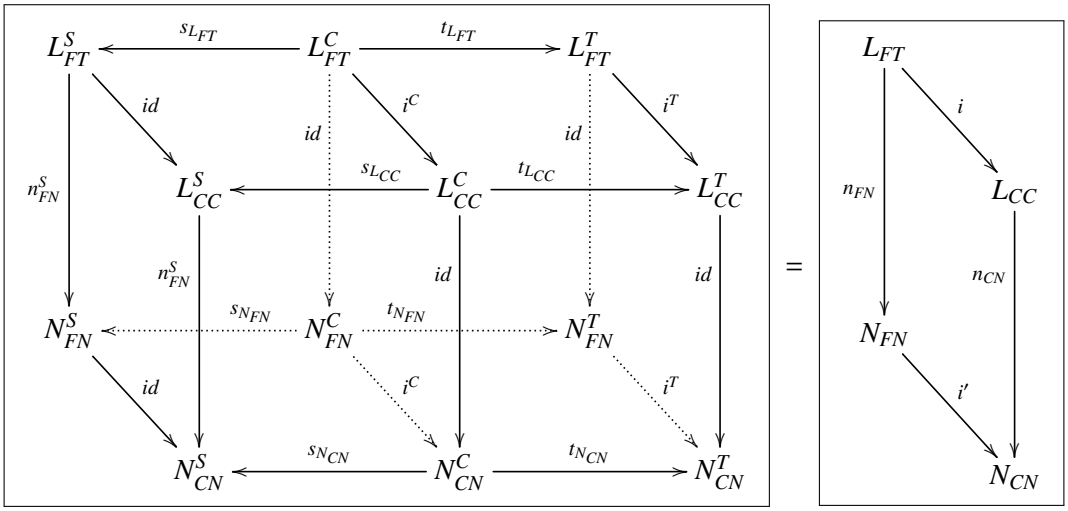
As presented in [13], some filter NACs can be generated automatically. In the case of our running example, the filter NAC for the backward translation rule Trainee2Trainee$_{BT}$ was generated by this technique.

In order to ensure that the synchronisation can be executed for any valid input and yields a correct result, we restrict our setting to filter NACs which are *domain specific*, i.e., which forbid structure either on the source or on the target component (domain). Formally, a NAC of a forward translation rule is

called domain specific, if it extends the LHS of the rule only on the source component. Symmetrically, a NAC of a backward translation rule is called domain specific, if it extends the LHS of the rule only on the target component. Extending the consistency creating rules TR_{CC} by propagating the domain specific filter NACs solves the problem of incompatibility between TR_{CC} and TR_{BT} and, moreover, does not introduce new incompatibilities with TR_{FT}.

	main components	NAC for each $n : L \to N$ of tr
tr_{CC}	$L_{CC} \xleftarrow{l_{CC}} K_{CC} \xrightarrow{r_{CC}} R_{CC}$ $\| \quad \| \quad \|$ $(R \oplus Att_L^{\mathbf{T}} \oplus Att_{R\setminus L}^{\mathbf{F}})\ (R \oplus Att_L^{\mathbf{T}})\ (R \oplus Att_L^{\mathbf{T}} \oplus Att_{R\setminus L}^{\mathbf{T}})$	$N_{CC} = (L_{CC} +_L N) \oplus Att_{N\setminus L}^{\mathbf{T}}$
tr_{FT}	$L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT}$ $\| \quad \| \quad \|$ $(L_F \oplus Att_{L_S}^{\mathbf{T}} \oplus Att_{R_S\setminus L_S}^{\mathbf{F}})\ (L_F \oplus Att_{L_S}^{\mathbf{T}})\ (R_F \oplus Att_{L_S}^{\mathbf{T}} \oplus Att_{R_S\setminus L_S}^{\mathbf{T}})$	$N_{FT} = (L_{FT} +_L N) \oplus Att_{N_S\setminus L_S}^{\mathbf{T}}$

Figure 17: Components of derived operational translation rules tr_{FT} and tr_{CC}



$n_{CN} = (n_{FN}^S, id, id)$, $N_{CN} = (N_{CN}^S \xleftarrow{s_{CN}} N_{CN}^T \xrightarrow{t_{CN}} N_{CN}^T)$, $s_{N_{CN}} = n_{FN}^S \circ s_{L_{CC}}$, $t_{N_{CN}} = t_{L_{CC}}$

Figure 18: Construction of propagated filter NAC $n_{CN} : L_{CC} \to N_{CN}$

Definition 4.5 (Filter NACs for CSynch)**.** Let tr be a triple rule, tr_{FT} be its derived forward translation, tr_{CC} its derived consistency creating rule and $i : L_{FT} \hookrightarrow L_{CC}$ be the corresponding inclusion of the left hand sides. Let $(n_{FN} : L_{FT} \hookrightarrow N_{FN})$ be a domain specific filter NAC, i.e., with $N_{FN} = (N_{FN}^S \leftarrow L_{FT}^C \rightarrow L_{FT}^T)$ as shown in Fig. 18. Then, the consistency creating rule with propagated filter NAC tr_{CN} extends tr_{CC} by the additional NAC $(n_{CN} : L_{CC} \to N_{CN})$ with: $N_{CN} = (N_{FN}^S \xleftarrow{s_{N_{CN}}} L_{CC}^C \xrightarrow{t_{N_{CN}}} L_{CC}^T)$, $n_{CN} = (n_{FN}^S, id, id)$, $s_{N_{CN}} = n_{FN}^S \circ s_{L_{CC}}$, and $t_{N_{CN}} = t_{L_{CC}}$. In the case of a backward translation rule, the construction is performed symmetrically. Let TGG be a triple graph grammar and TR_{FN} and TR_{BN} be the derived sets of forward and backward translation rules possibly extended by filter NACs. Then, $CSynch(TGG, \mathsf{CSynch}_{FN})$ is

derived by extending the consistency creating rules TR_{CC} with all propagated filter NACs from TR_{FN} and TR_{BN} leading to a set TR_{CN} and performing the construction as for $CSynch(TGG, \mathsf{CSynch})$ on these sets of extended rules $(TR_{CN}, TR_{FN}, TR_{BN})$.

In our second main technical result in Thm. 2 below, we show that the introduction and propagation of filter NACs does not affect the formal properties of correctness and completeness for the non-deterministic concurrent synchronization framework.

By Thm. 1 in [15, 18] with formal proofs in [17], we know that the derived synchronization framework of a TGG with deterministic sets of operational rules $(TR_{CC}, TR_{FT}, TR_{BT})$ is correct, complete and invertible. In the present paper, the only difference is that the operational rules may be extended with additional filter NACs. We need to extend the original proof to show that the additional filter NACs do not affect the correctness property.

Theorem 2 (Correctness of Concurrent Synchronization Frameworks with Efficiency Improvement by Filter NACs)**.** Given a triple graph grammar TGG and a set of domain specific filter NACs for the operational translation rules that have been propagated to the consistency creating rules TR_{CC}. Then, the derived non-deterministic concurrent synchronisation framework with domain specific filter NACs $\mathsf{CSync}(TGG, \mathsf{CSynch}_{FN})$ is correct and complete.

We will use the follwing fact (for the proof see Fact 10 in [17]) to show Fact 4.2, which provides the first step for the proof of Thm. 2.

Fact 4.1 (Equivalence of Triple and Extended Consistency Creating Sequences)**.** Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar with derived consistency creating rules TR_{CC} and given $G \in VL(TG)$. Then, the following are equivalent for almost injective matches

1. There is a TGT-sequence $s = (\emptyset \overset{tr^*}{\Longrightarrow} G_k)$ via TR with injective embedding $f : G_k \to G$.
2. There is a consistency creating sequence $s' = (G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_k)$ via TR_{CC} with $G'_0 = Att^{\mathbf{F}}(G)$.

Moreover, the sequences correspond via $G'_k = H \oplus Att^{\mathbf{T}}_{G_k} \oplus Att^{\mathbf{F}}_{H \setminus G_k}$.

Fact 4.2 (Induced Triple Sequence of Consistency Creating Sequence with additinal NACs)**.** Let $TGG = (TG, \emptyset, TR)$ be a triple graph grammar with derived consistency creating rules TR_{CC} and let TR'_{CC} be obtained from TR_{CC} by possibly adding some NACs to some of the rules. Then, each consistency creating sequence $s' = (G'_0 \overset{tr^*_{CC}}{\Longrightarrow} G'_k)$ via TR'_{CC} with $G'_0 = Att^{\mathbf{F}}(G)$ induces a triple sequence $s = (\emptyset \overset{tr^*}{\Longrightarrow} G_k)$ via TR with $G_k \subseteq G$.

Proof. Consistency creating sequence s' via TR'_{CC} is also a consistency creating sequence via TR_{CC}, because disregarding NACs the sets TR'_{CC} and TR_{CC} are the same and all NACs of a rule tr_{CC} in TR_{CC} are also NACs of the corresponding rule tr'_{CC} in TR'_{CC}. Thus, we can apply Fact 4.1 to the sequence s' via TR_{CC} and derive the corresponding triple sequence $s = (\emptyset \overset{tr^*}{\Longrightarrow} G_k)$ via TR. □

Fact 4.2 provides the basis for the following Fact 4.3, which we use in the proof of Thm. 2.

Fact 4.3 (Consistency Creating Sequence with Propagated Filter NACs and Induced Forward Translation sequence with Filter NACs)**.** Let TGG be a triple graph grammar and TR_{FN} and TR_{BN} be the derived sets of forward and backward translation rules possibly extended by domain specific filter NACs, and let TR_{CN} be the set of consistency creating rules derived from TR_{CC} by propagating all filter NACs of TR_{FN} and TR_{BN}.

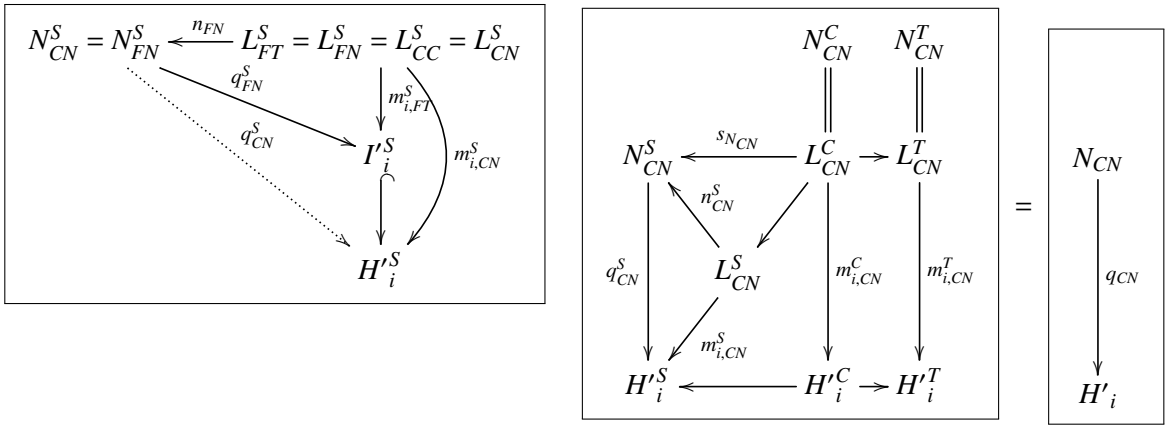


Figure 19: Constructions for the proof of Fact 4.3

Let $s = (H'_0 \overset{tr^*_{CN}}{\Longrightarrow} H'_k)$ via TR_{CN} with $H'_0 = Att^{\mathbf{F}}(H)$. Then, there is a triple graph $G \subseteq H$ and a forward translation sequence $s_{FN} = (I'_0 \overset{tr^*_{FN}}{\Longrightarrow} I'_k)$ via TR_{FN}, such that $I'_0 = (H'^S_0 \leftarrow \varnothing \rightarrow \varnothing)$ and $I'_k = (H'^S_k \leftarrow G^C \rightarrow G^T)$.

Proof. Consistency creating sequence $s = (H'_0 \overset{tr^*_{CN}}{\Longrightarrow} H'_k)$ via TR_{CN} with $H'_0 = Att^{\mathbf{F}}(H)$ implies a corresponding triple sequence $s = (\varnothing \overset{tr^*}{\Longrightarrow} G)$ via TR by Fact 4.2. By the composition and decomposition result for TGGs with application conditions (Thm. 1 in [8]) this implies a source consistent forward sequence $s_F = (G_0 \overset{tr^*_F}{\Longrightarrow} G_k)$ via TR_F with $G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$. By Fact 1 in [14] (Equivalence of complete forward translation sequences with source consistent forward sequences), there is a corresponding forward translation sequence $s_{FT} = (G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_k)$ via TR_{FT} with $G'_0 = (Att_{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ and $G'_k = (Att_{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.

By Fact 6 in [17] (Extension of FT-sequences), this sequence implies the existence of the corresponding extended sequence via forward translation rules $s'_{FT} = (I'_0 \overset{tr^*_{FT}}{\Longrightarrow} I'_k)$ via TR_{FT} with $I'_0 = (Att_{\mathbf{F}}(H^S) \leftarrow \varnothing \rightarrow \varnothing)$ and $I'_k = (H^S \oplus Att^{\mathbf{T}}_{G^S} \oplus Att^{\mathbf{F}}_{H^S \setminus G^S} \leftarrow G^C \rightarrow G^T)$.

It remains to show that sequence s'_{FT} satisfies in each step also the additional filter NACs in TR_{FN}. We show this by contraposition. Assume that there is a domain specific filter NAC $(n_{FN} \colon L_{i,FN} \rightarrow N_{FN})$ of rule $tr_{i,FN}$ and there is a step $I'_i \overset{tr_{i,FT}, m_{i,FT}}{\Longrightarrow} I'_{i+1}$ not satisfying NAC n_{FN}. This means that there is an almost injective morphism $q_{FN} \colon N_{FN} \rightarrow I'_i$ compatible with $m_{i,FT}$, i.e., $q_{FN} \circ n_{FN} = m_{i,FT}$.

We extend morphism q_{FN} as shown in the two diagrams above. On the source component, we obtain $q^S_{CN} = i^S \circ q^S_{FN}$ using the inclusion $i \colon I_i \hookrightarrow H_i$. On the correspondence and target components we can take the match $m_{i,CN}$, because the NAC is domain specific concerning the source domain.

Morphism q_{CN} is a triple graph morphism, because all diagrams above commute. It is almost injective, because q_{FN} is almost injective and $i \colon I_i \hookrightarrow H_i$ is an inclusion. Finally, $q_{CN} \circ n_{CN} = m_{i,CN}$ as depicted in the diagrams above, where the source component is depicted right and commutativity on the correspondence and target component holds by: $q^C_{CN} \circ n^C_{CN} = m^C_{CN} \circ id = m^C_{CN}$ and $q^T_{CN} \circ n^T_{CN} = m^T_{CN} \circ id = m^T_{CN}$. Thus, morphism $q_{CN} \colon N_{CN} \rightarrow H_i$ violates a NAC of the step $H_i \overset{tr_{i,CN} H_{i+1}}{\Longrightarrow}$ that corresponds to forward translation step $s_{FN} = (I'_0 \overset{tr^*_{FN}}{\Longrightarrow} I'_k)$. This is a contraction to the given NAC-consistent consistency creat-

ing sequence $s = (H'_0 \overset{tr^*_{CN}}{\Longrightarrow} H'_k)$ via TR_{CN}. Therefore, the assumption that there is a forward translation step is invalid. Thus, forward translation sequence $s'_{FT} = (I'_0 \overset{tr^*_{FT}}{\Longrightarrow} I'_k)$ via TR_{FT} is NAC consistent for all NACs of TR_{FN} and therefore, it is a consistent sequence via TR_{FN}. □

Proof of Thm. 2. Operations fAln and bAln are defined for all inputs, because they are based on pullback constructions. Operation Del is given by a terminating execution of a consistency creating sequence via TR_{CN}. This condition is ensured by the precondition that the set TR_{CN} ensures termination. Finally, operations fAdd and bAadd are not ensured to yield the required results for all possible inputs, because the additional NACs may cut off some transformation seuqences, i.e. they become shorter. Thus, we need to show that the composed operation fPpg is still defined for all inputs and the resulting output is as required.

From the computed consistency creating sequence via operation Del, we derive the corresponding forward translation sequence $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_k$ via TR_{FN} using Fact 4.3. This sequence can be extended to a terminated forward translation sequence $G'_0 \overset{tr^*_{FN}}{\Longrightarrow} G'_k \overset{tr'^*_{FN}}{\Longrightarrow} G'_n$ via TR_{FN}. In the general case, we may have to backtrack till we derive a complete forward translation sequence. But, due to the completeness result for forward translation sequences with filter NACs [13], we know that there is at least one such sequence. This ensures completeness for the synchronisation operation using the completeness result for concurrent synchronisation without filter NACs [12].

By Def. 6 in [13], a model transformation based on forward translation rules is based on complete forward translation sequences. Thus, by Thm. 1 in [13] (correctness), we can conclude that $G = (G^S \leftarrow B^C \rightarrow G^T) \in VL$. Therefore, operation fPpg is correct and symmetrically, we derive that operation bPpg is correct. This implies that the concurrent synchronization operation CSynch is correct as well. □

Note that if the sets TR_{CN} and TR_{FN} do not require backtracking, we derive by the above proof that we do not need to perform backtracking for the forward propagation operation. Symmetrically, if the sets TR_{CN} and TR_{FN} do not require backtracking, we derive by the above proof that we do not need to perform backtracking for the backward propagation operation. As presented inčiteHEGO10, this condition can be checked automatically using the tool AGG [27].

As shown by Thm. 2 above, the extension of the synchronisation operations with filter NACs does not affect the correctness and completeness results for the derived synchronisation framework. This ensures that concurrent model synchronisation can be performed efficiently also in cases where the TGG operations initially require backtracking, but do not require backtracking using the generated filter NACs.

5 Domain Priorisation

After introducing the theory for concurrent model synchronisation with conflict resolution in Sec. 3 and Sec. 4, this section illustrates the application scenario for concurrent model synchronisation from Sec. 2 in more detail.

The theory addresses the problem of synchronising two interrelated models after both models were changed concurrently by a model modification, respectively. Concurrent model modifications may lead to conflicts which need to be resolved by the model synchronisation. Two synchronisation methods are considered: (1) The modification of the source model is forward propagated to the target domain first leading to a merged modification of the target model where conflicts are resolved, then the merged target modification is backward propagated to the source domain leading to a modification of the source model, (2) The modification of the target model is backward propagated to the source domain first leading to a merged modification of the source model where conflicts are resolved, then the merged source modification is forward propagated to the target domain leading to a modification of the target model. This induces a priorisation of the source and target domains in view of conflict resolution, i.e., by forward propagating first, the conflict resolution is performed in the target domain, whereas by backward propagating first, the conflict resolution is performed in the source domain.

The application scenario comprises the administration of a fictive company's personnel. The company consists of a development, a marketing and an administration department. The administration department is responsible for administering the personnel data of all departments, i.e., add new personnel, change the details of the existing personnel and delete personnel who leaves the company in the electronic administration system. The marketing department additionally administers the data of its personnel independently from the administration department. Therefore, the administration and the marketing department administer their own models of staff data, respectively, so that both models may be modified concurrently. After concurrent modification, both models need to be synchronised to be consistent and conflicts need to be resolved.

Each of the examples 5.1 to 5.16 contain a concurrent modification of both models and illustrates the first two steps of their synchronisation twofold, firstly, by starting with forward propagating the modification of the marketing department to the administration department, and secondly by starting with backward propagating the modification of the administration department to the marketing department, respectively. Consider that forward propagating first can lead to conflicts (or no conflicts, respectively), whereas backward propagating first may not, as illustrated by Ex. 5.14 and Ex. 5.16. This reveals an interesting property of model synchronisations, that changing the order of the forward and backward propagation steps, therefore, giving the source domain priority over the target domain or vice versa with regard to conflict resolution, may lead to different synchronisation results. The remaining steps necessary for the complete synchronisation of each example are treated in the following Sec. 6.

In all examples, the modifications made to the models induce already consistent models so that the consistency creating steps $1 : CCS$ and $1 : CCT$ always lead to identical models with d_F^S and d_B^T being identity morphisms so that $G_1^S = G_{1,C}^S, G_1^T = G_{1,C}^T, d_1^S = d_{1,CC}^S$ and $d_1^T = d_{1,CC}^T$.

Example 5.1 (Case 1 - Appointment of Molly Murphy)**.** Molly Murphy is appointed as a new member of the marketing department. Therefore, In Fig. 20 and Fig. 21, she is added to the marketing department by the administration department with modification d_1^T but not by the marketing department itself indicated by the identity modification d_1^S. In Fig. 20, the non-changing modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to an identity modification $d_{1,F}^T$ which reflects the non-changing nature of d_1^S in the target domain. Conversely, in Fig. 21, modification d_1^T with

$d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the addition of Molly Murphy in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict.

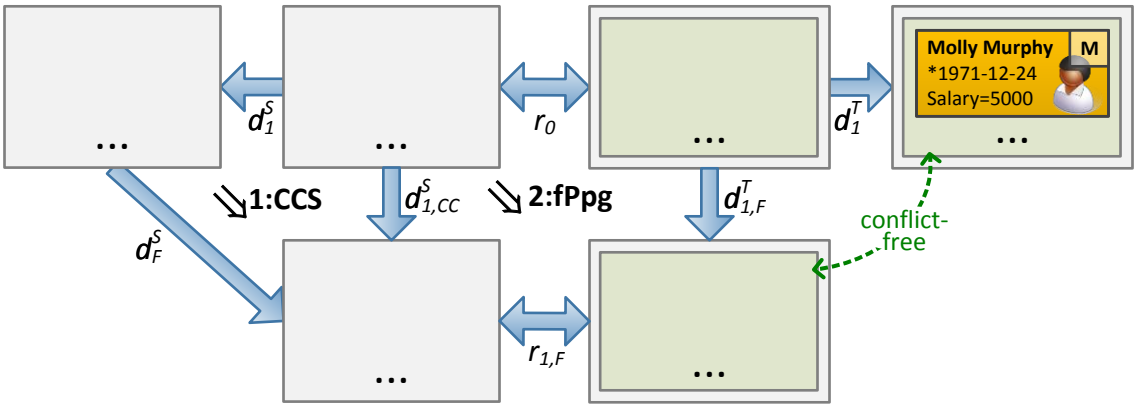


Figure 20: Appointment of Molly Murphy - $fPpg$ first

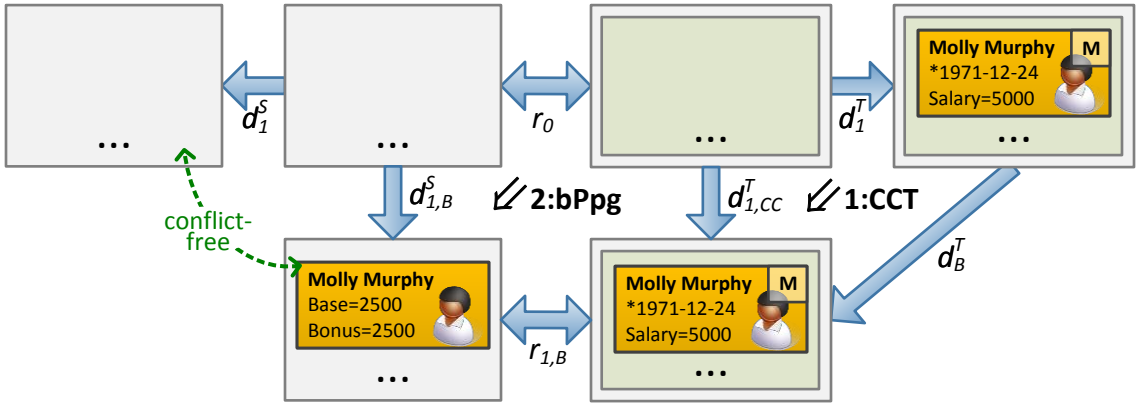


Figure 21: Appointment of Molly Murphy - $bPpg$ first

Example 5.2 (Case 2 - Appointment of Harrison Hunt)**.** Harrison Hunt is appointed as a new member of the marketing department. Therefore, in Fig. 22 and Fig. 23, he is added but in the reverse order to Ex. 5.1, i.e., he is added by the marketing department with modification d_1^S but not by the administration department indicated by the identity modification d_1^T. In Fig. 22, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the addition of Harrison Hunt in the target domain. Conversely, in Fig. 23, the non-changing modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to an identity modification $d_{1,B}^S$ which reflects the non-changing nature of d_1^T in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict, respectively.

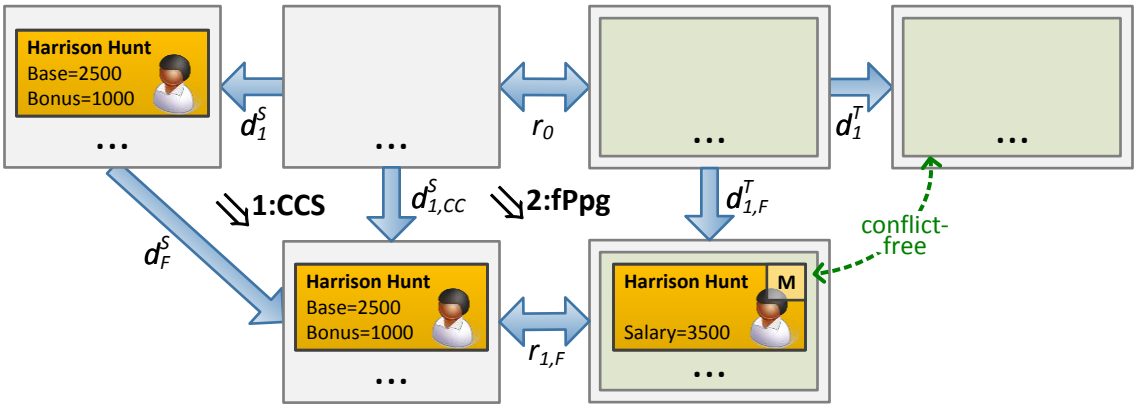


Figure 22: Appointment of Harrison Hunt - $fPpg$ first

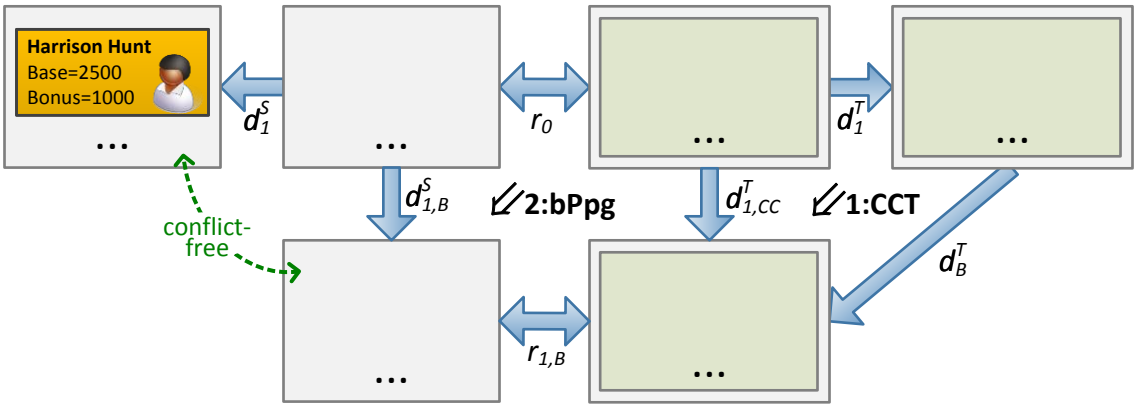


Figure 23: Appointment of Harrison Hunt - $bPpg$ first

Example 5.3 (Case 3 - Resignation of Paul Page)**.** Paul Page has quit his job. Therefore, in Fig. 24 and Fig. 25, Paul Page as a former member of the marketing department is deleted by the administration department with modification d_1^T but not by the marketing department itself indicated by the identity modification d_1^S. In Fig. 24, the non-changing modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to an identity modification $d_{1,F}^T$ which reflects the non-changing nature of d_1^S in the target domain. Conversely, in Fig. 25, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the deletion of Paul Page in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict, respectively.

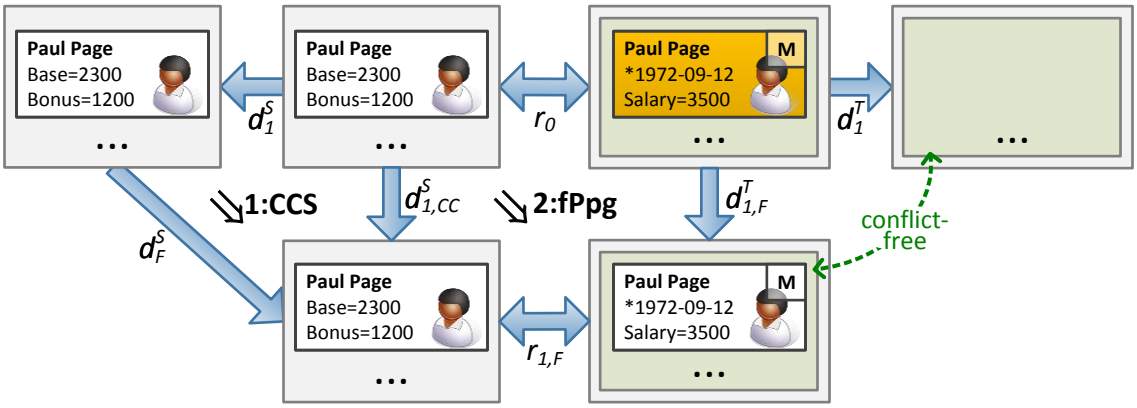


Figure 24: Resignation of Paul Page - $fPpg$ first

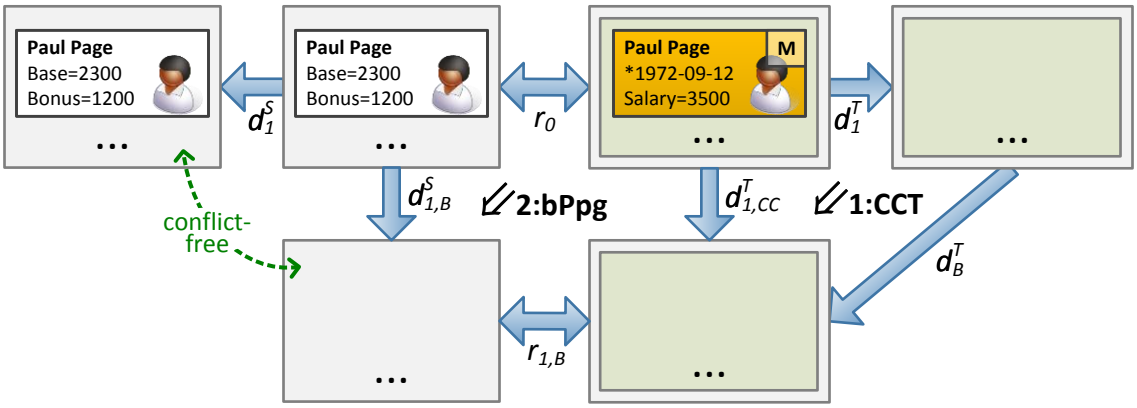


Figure 25: Resignation of Paul Page - $bPpg$ first

Example 5.4 (Case 4 - Transfer of Henry Hunter)**.** Henry Hunter is transferred from the marketing to the development department. Therefore, in Fig. 26 and Fig. 27, the affiliation status of Henry Hunter is changed from M to D by the administration department with modification d_1^T but not by the marketing department indicated by the identity modification d_1^S. In Fig. 26, the non-changing modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to an identity modification $d_{1,F}^T$ which reflects the non-changing nature of d_1^S in the target domain. Conversely, in Fig. 27, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the change of Henry's status in the source domain, i.e., since in the source domain only personnel of the marketing department are administered, Henry Hunter is deleted due to his transfer to the development department. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict, respectively.

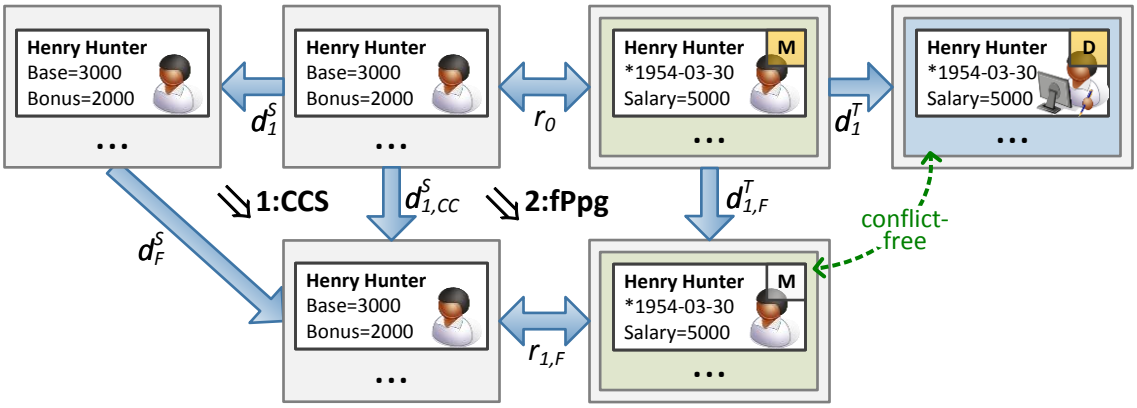


Figure 26: Transfer of Henry Hunter - $fPpg$ first

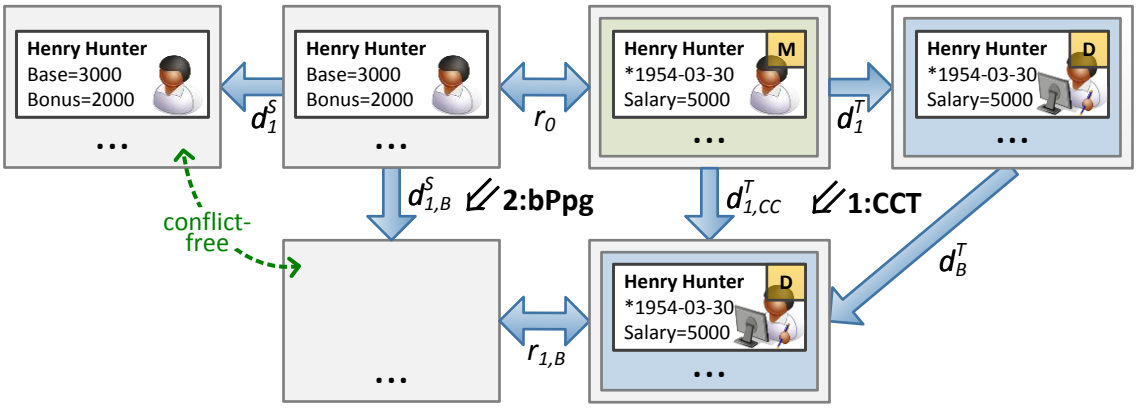


Figure 27: Transfer of Henry Hunter - $bPpg$ first

Example 5.5 (Case 5 - Resignation of Lucy Lewis)**.** Lucy Lewis will leave the company. Therefore, in Fig. 28 and Fig. 29, Lucy Lewis as a former member of the marketing department is deleted by the marketing department with modification d_1^S but not by the administration department indicated by the identity modification d_1^T. In Fig. 28, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the deletion of Lucy Lewis in the target domain. Conversely, in Fig. 29, the non-changing modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to an identity modification $d_{1,B}^S$ which reflects the non-changing nature of d_1^T in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict, respectively.

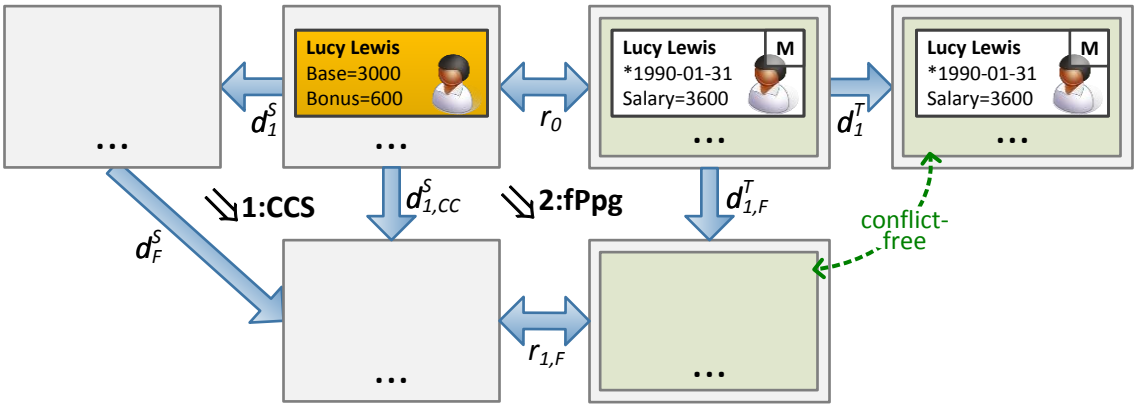


Figure 28: Resignation of Lucy Lewis - $fPpg$ first

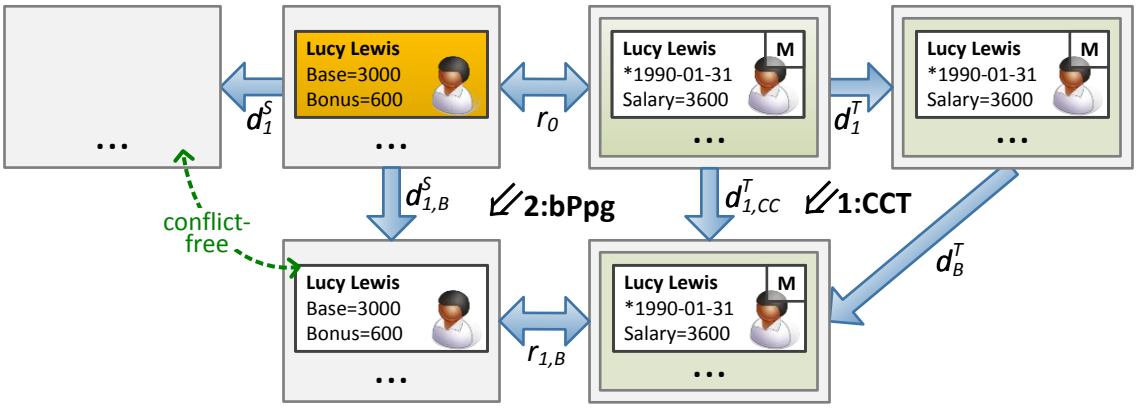


Figure 29: Resignation of Lucy Lewis - $bPpg$ first

Example 5.6 (Case 6 - Renaming of K.-A. Knight)**.** The marketing department detects that the first name of *Kai-Axl Knight* is abbreviated in the model. Therefore, in Fig. 30 and Fig. 31, the first name is changed from the abbreviation *K.-A.* to *Kai-Axl* by the marketing department with modification d_1^S but not by the administration department indicated by the identity modification d_1^T. In Fig. 30, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the change of the first name in the target domain. Conversely, in Fig. 31, the non-changing modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the non-changing nature of d_1^T in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict, respectively.

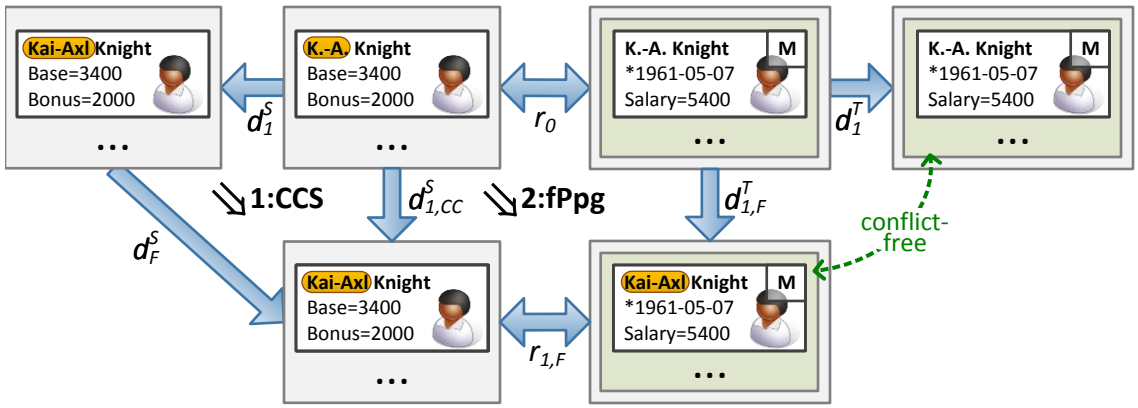


Figure 30: Renaming of K.-A. Knight - $fPpg$ first

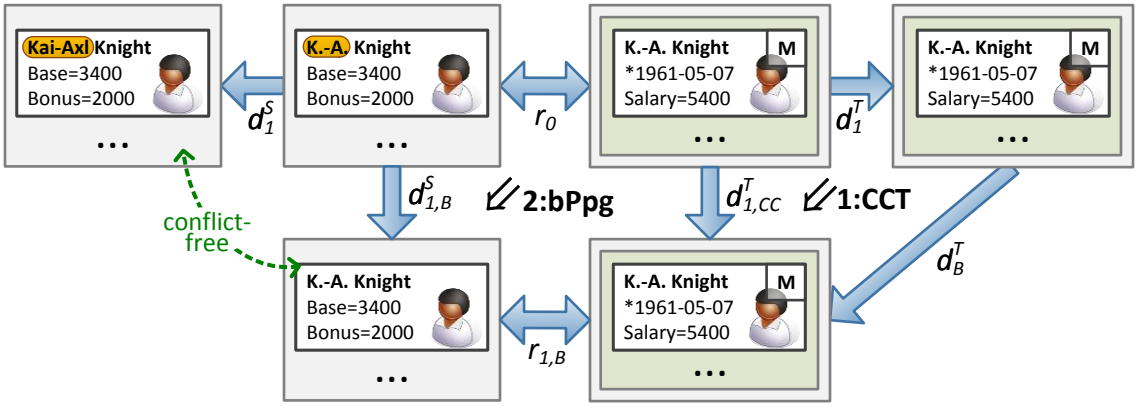


Figure 31: Renaming of K.-A. Knight - $bPpg$ first

Example 5.7 (Case 7 - Appointment of Max McDonald)**.** Max McDonald is appointed as a new member of the marketing department. Therefore, in Fig. 32 and Fig. 33, he is concurrently added by the marketing and the administration department with modifications d_1^S or d_1^T, respectively. In Fig. 32, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the addition of Max McDonald in the target domain. Conversely, in Fig. 33, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the addition of Max McDonald in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are not in conflict, respectively.

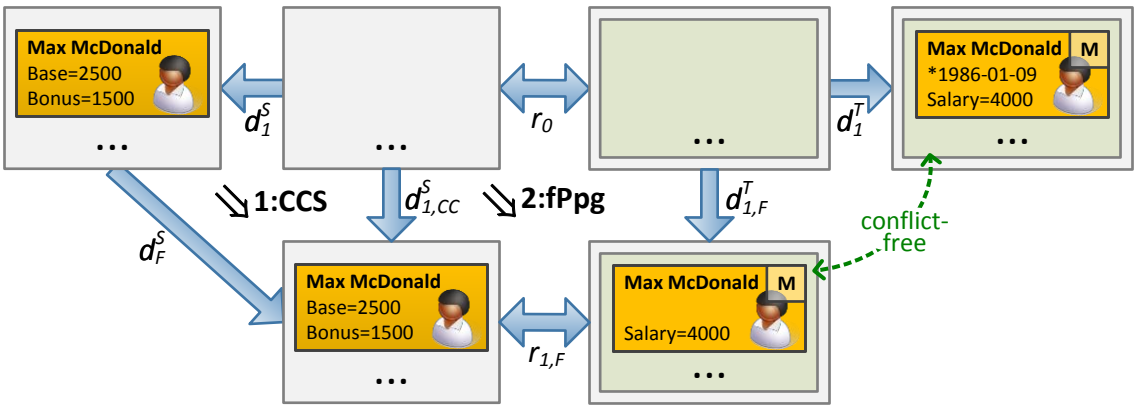


Figure 32: Appointment of Max McDonald - *fPpg* first

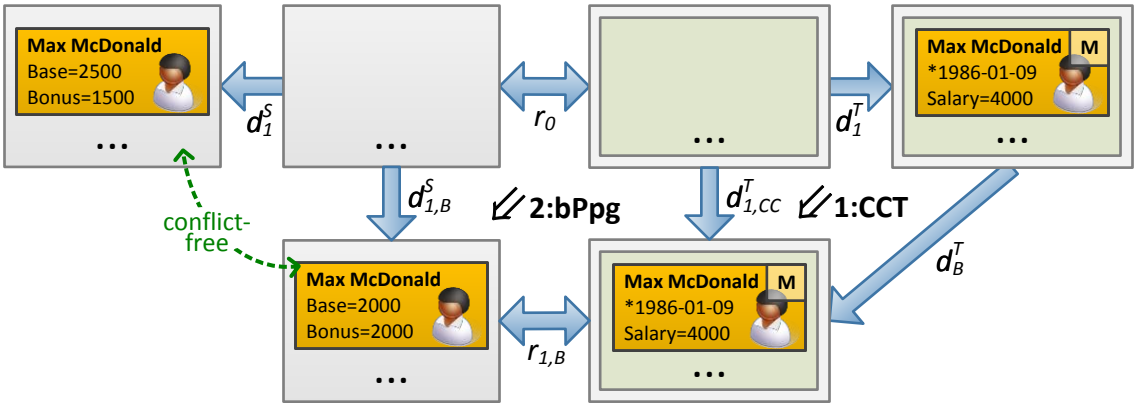


Figure 33: Appointment of Max McDonald - *bPpg* first

Example 5.8 (Case 8 - Retirement of Daniel Davis)**.** Daniel Davis has retired. Therefore, in Fig. 34 and Fig. 35, Daniel Davis as a former member of the marketing department is concurrently deleted by the marketing and the administration department with modifications d_1^S or d_1^T, respectively. In Fig. 34, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the deletion of Daniel Davis in the target domain. Conversely, in Fig. 35, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the deletion of Daniel Davis in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are in delete-delete conflict, respectively.

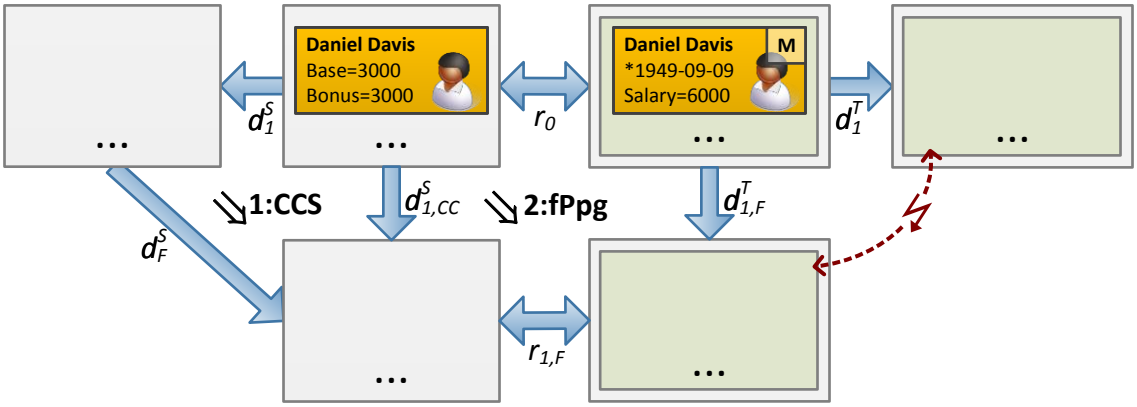


Figure 34: Retirement of Daniel Davis - $fPpg$ first

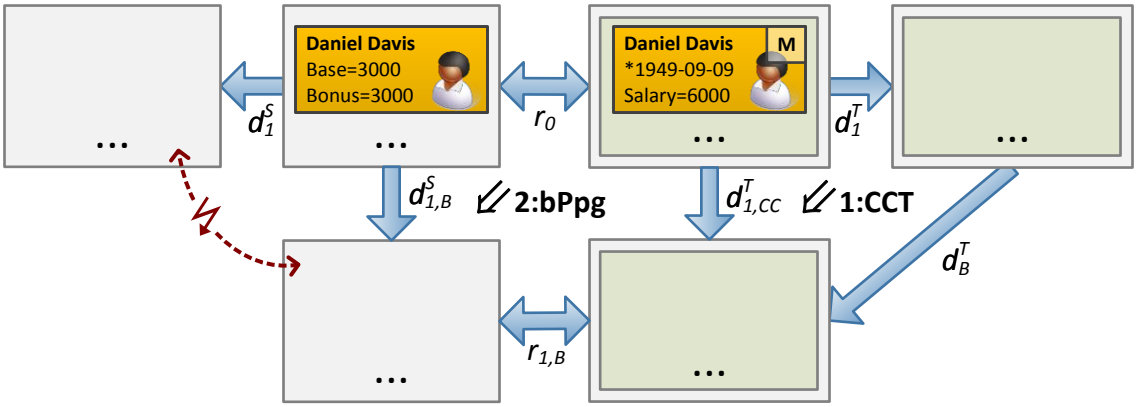


Figure 35: Retirement of Daniel Davis - $bPpg$ first

Example 5.9 (Case 9 - Transfer of Freya Fisher)**.** Freya Fisher is transferred from the marketing to the development department. Therefore, in Fig. 36 and Fig. 37, she is deleted by the marketing department with modification d_1^S and concurrently to that her affiliation status is changed from M to D by the administration department with modification d_1^T. In Fig. 36, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the deletion of Freya Fisher in the target domain. Conversely, in Fig. 37, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the change of Freya's status in the source domain, i.e., since in the source domain only personnel of the marketing department are administered, Freya Fisher is deleted due to her transfer to the development department. The modifications d_1^T and $d_{1,F}^T$ are in delete-use conflict, i.e., the node of Freya Fisher is deleted by $d_{1,F}^T$ but is used by d_1^T to change the affiliation status. Furthermore, modifications d_1^S and $d_{1,B}^S$ are in delete-delete conflict.

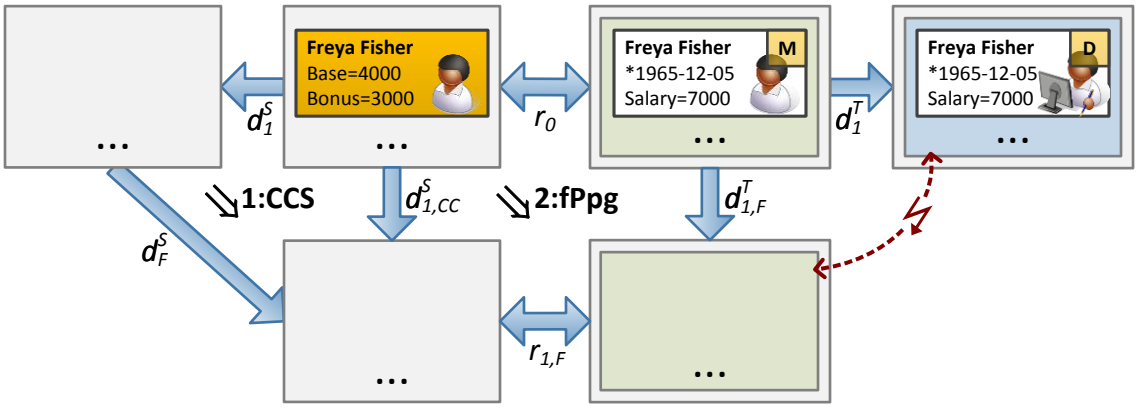


Figure 36: Transfer of Freya Fisher - $fPpg$ first

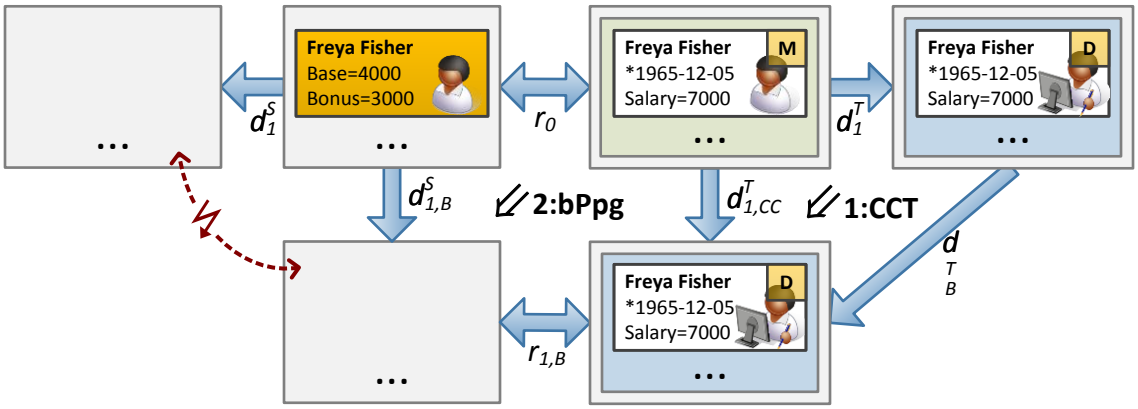


Figure 37: Transfer of Freya Fisher - $bPpg$ first

Example 5.10 (Case 10 - Renaming of Holly J. Hill)**.** The marketing and the administration department simultaneously detect that the first name of *Holly J. Hill* is abbreviated in the model, respectively. Therefore, in Fig. 38 and Fig. 39, the first name is changed from the abbreviation *Holly J.* to *Holly Jane* concurrently by the marketing and the administration department with modification d_1^S or d_1^T, respectively. In Fig. 38, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the change of the first name in the target domain. Conversely, in Fig. 39, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the change of the first name in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are in delete-delete conflict, respectively, i.e., concurrently changing the abbreviated first name, requires its concurrent deletion first.

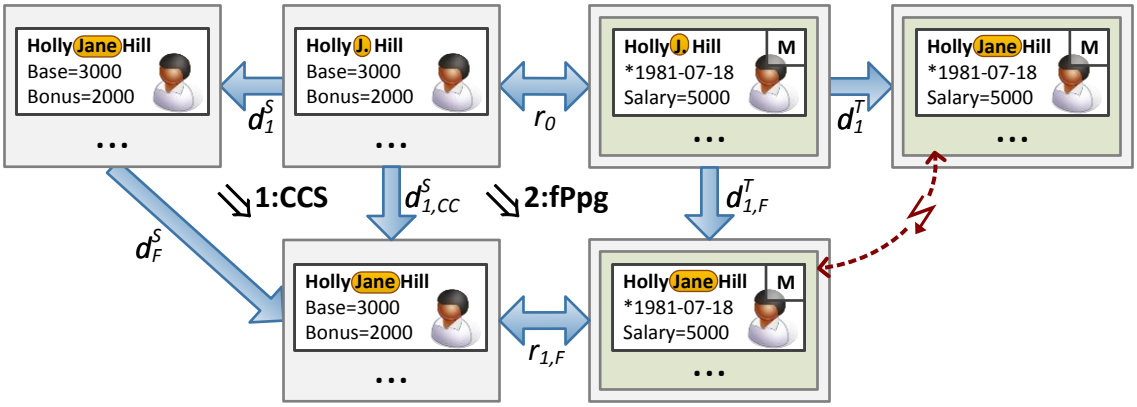


Figure 38: Renaming of Holly J. Hill - $fPpg$ first

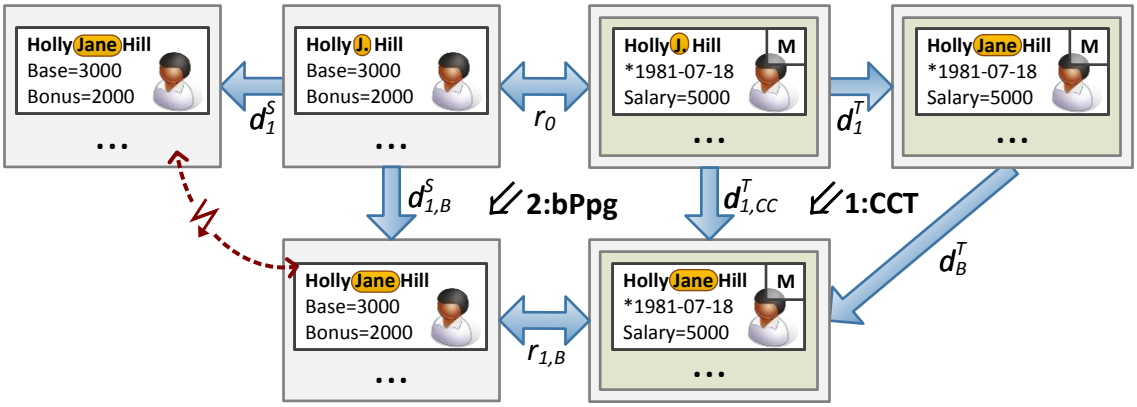


Figure 39: Renaming of Holly J. Hill - $bPpg$ first

Example 5.11 (Case 11 - Renaming of Mia Miller)**.** The marketing and the administration department simultaneously detect that the last name of *Mia Miller* is not valid any more due to her marriage. Therefore, in Fig. 40 and Fig. 41, the last name is changed concurrently from *Miller* to *MacDonald* by the marketing department with modification d_1^S and from *Miller* to *McDonald* by the administration department with modification d_1^T. The last name is changed differently by both departments due to a typo made by the administration department. In Fig. 40, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the change of the last name in the target domain. Conversely, in Fig. 41, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the change of the last name in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are in delete-delete conflict, respectively, i.e., concurrently changing the last name, requires its concurrent deletion first.

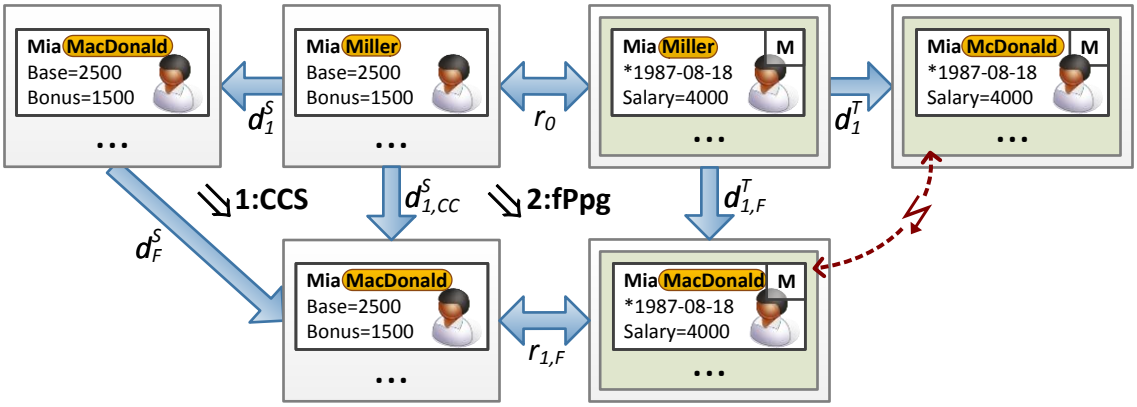


Figure 40: Renaming of Mia Miller - $fPpg$ first

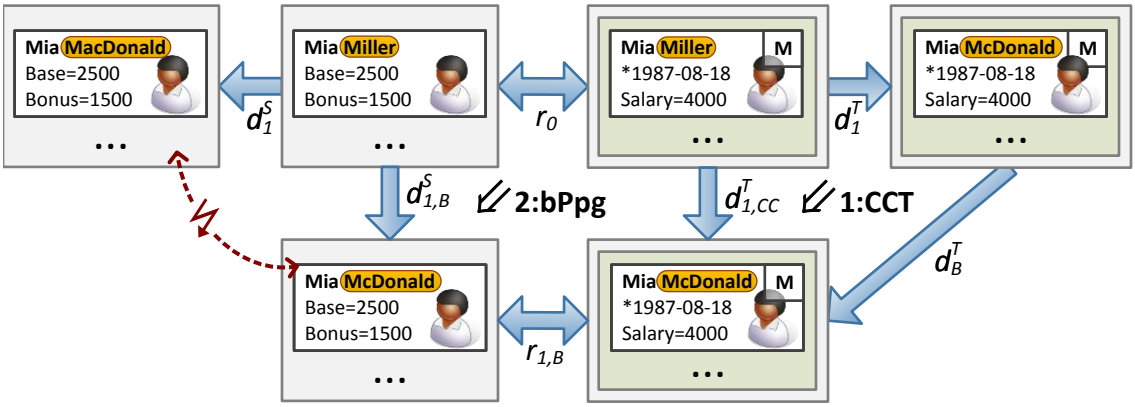


Figure 41: Renaming of Mia Miller - $bPpg$ first

Example 5.12 (Case 12 - Changing the salary and bonus of Alex Archer)**.** Alex Archer as a long-standing member of the marketing department performed particularly well during the last years. Therefore, in Fig. 42 and Fig. 43, his bonus is increased from *1000* to *1500* by the marketing department with modification d_1^S due to his great performance and concurrently to that, in addition the salary is increased from *4000* to *4050* by the administration department with modification d_1^T due to an inflationary adjustment. In Fig. 42, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the increase of the bonus by increasing the salary from *4000* to *4500* in the target domain. Conversely, in Fig. 43, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the increase of the salary by adapting the base and bonus in the source domain. Consider that by changing the salary in the target domain, triple rule *DetailedSalary2Salary* of Fig. 5 ensures values for base and bonus so that $Base + Bonus = Salary$. Since it is not clear how to divide the salary value to base and bonus values, a strategy needs to be applied to obtain a unique result in the backward propagation step. For the presented application scenario it holds that $Salary = 2 \cdot Base = 2 \cdot Bonus$. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are in delete-delete conflict, respectively, i.e., concurrently changing the salary or bonus, requires its concurrent deletion first, respectively.

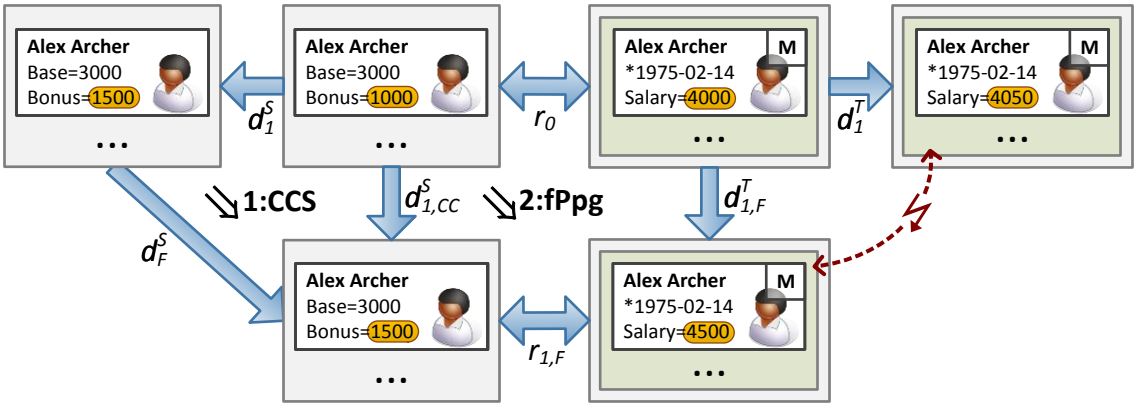


Figure 42: Changing the salary and bonus of Alex Archer - $fPpg$ first

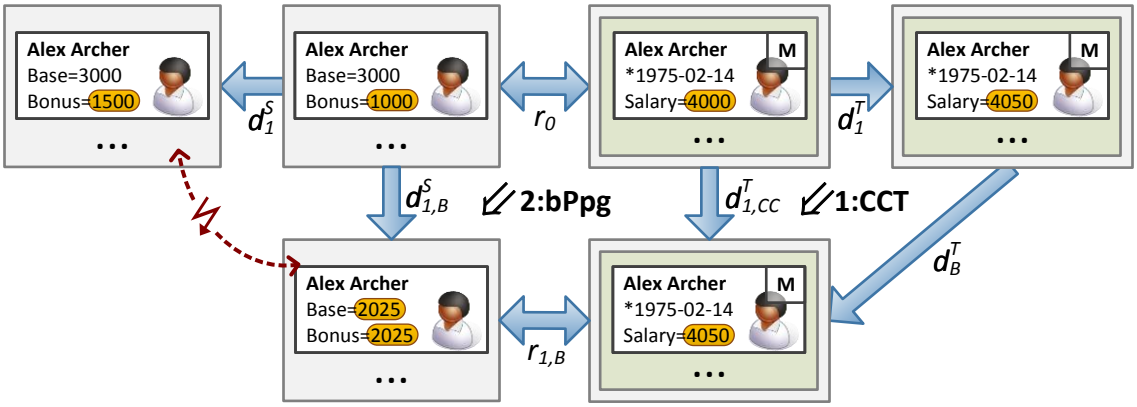


Figure 43: Changing the salaray and bonus of Alex Archer - $bPpg$ first

Example 5.13 (Case 13 - Resignation and changing the base salary of Jack Judge)**.** Jack Judge as a long-standing member of the marketing department will quit his job next month. A rule that is specific to the marketing department includes an increase of the base salary after every two years. Therefore, in Fig. 44 and Fig. 45, his base salary is increased from *2000* to *3000* by the marketing department with modification d_1^S due to his long-time affiliation but concurrently to that, he is already deleted by the administration department with modification d_1^T due to his early resignation. In Fig. 44, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the increase of the base salary by increasing the salary from *4000* to 5000 in the target domain. Conversely, in Fig. 45, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the deletion of Jack Judge in the source domain. The modifications $d_1^T, d_{1,F}^T$ and $d_1^S, d_{1,B}^S$ are in delete-use conflict, respectively, i.e., the node of Jack Judge is deleted while it is concurrently used to change the base or overall salary, respectively.

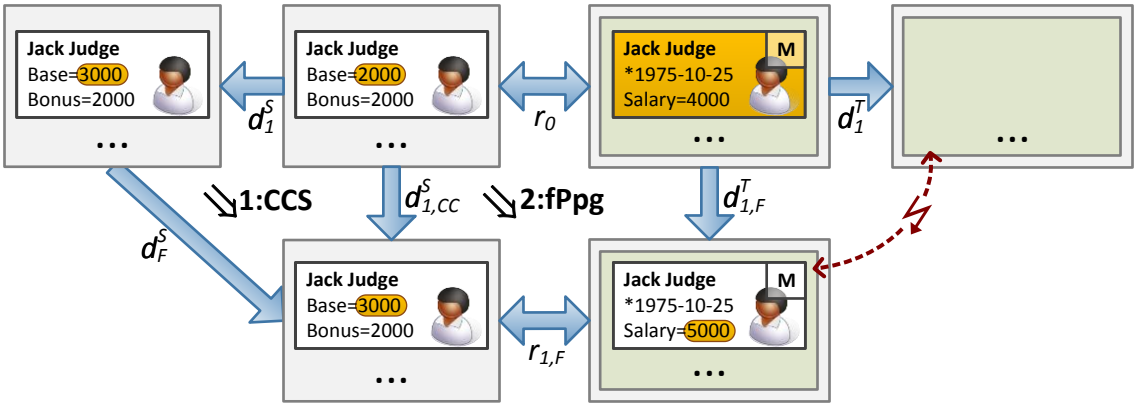


Figure 44: Resignation and changing the base salary of Jack Judge - $fPpg$ first

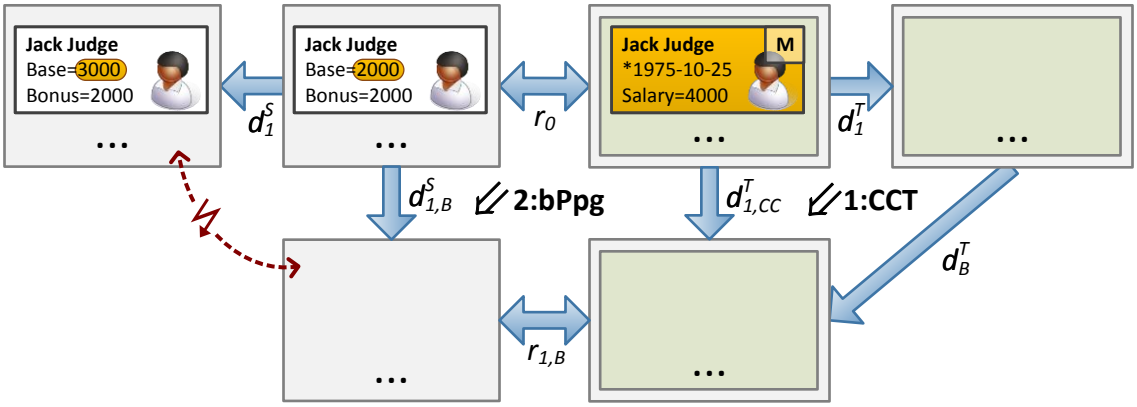


Figure 45: Resignation and changing the base salary of Jack Judge - $bPpg$ first

Example 5.14 (Case 14 - Resignation and changing the birth date of Tony Taylor)**.** Tony Taylor as a member of the marketing department will leave the company next month. At the same time, another Tony Taylor will start at the same department. This situation leads to misunderstandings where the administration department mistakenly modifies the data of the old Tony Taylor to adopt them to those of the new Tony. Therefore, in Fig. 46 and Fig. 47, Tony Taylor is deleted by the marketing department with modification d_1^S due to his early resignation but concurrently to that, his birth date is mistakenly adapted to the birth date of the new Tony Taylor by the administration department with modification d_1^T. In Fig. 46, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the deletion of Tony Taylor in the target domain. Conversely, in Fig. 47, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to identity modification $d_{1,B}^S$ which does not reflect the change of the birth date in the source domain as this information is not available in the source domain. The modifications $d_1^T, d_{1,F}^T$ are in delete-use conflict, i.e., the node of Tony Taylor is deleted while it is concurrently used to change the birth date, but modifications $d_1^S, d_{1,B}^S$ are conflict-free. This leads to an interesting property of model synchronisations, that a different order of forward and backward propagation steps may lead to different synchronisation results. Ex. 6.14 in Sec. 6 illustrates the remaining steps necessary for the complete synchronisation of this example and shows the different results that are obtained by switching the order

of forward and backward propagation steps.

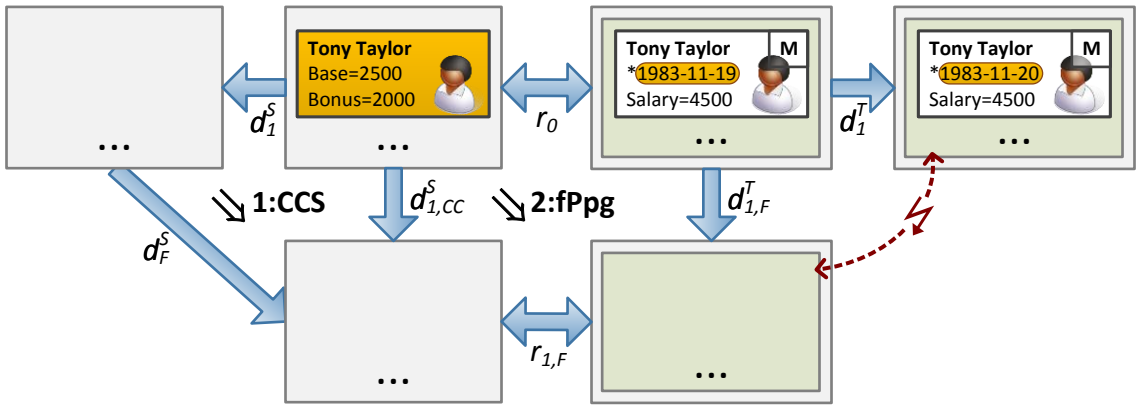


Figure 46: Resignation and changing the birth date of Tony Taylor - $fPpg$ first

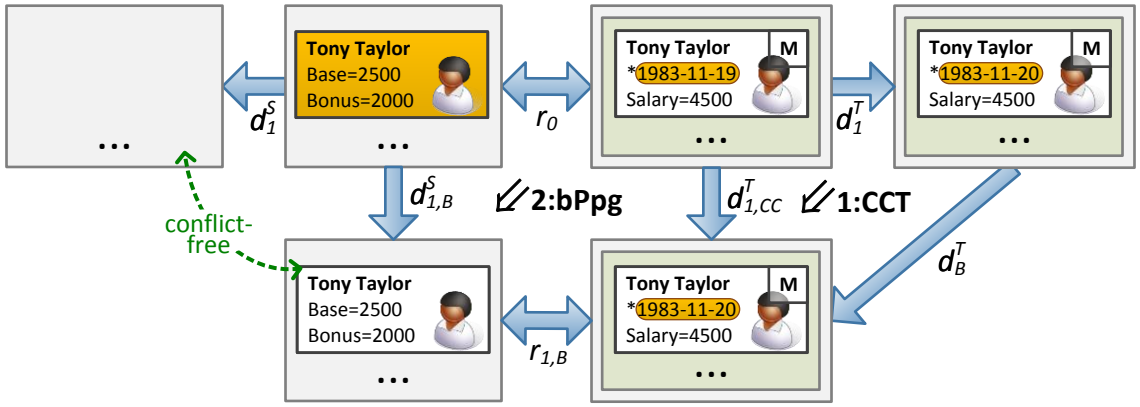


Figure 47: Resignation and changing the birth date of Tony Taylor - $bPpg$ first

Example 5.15 (Case 15 - Transfer of Lily Lee)**.** Lily Lee as a trainee of the marketing department is transferred to the development department. Therefore, in Fig. 48 and Fig. 49, she is deleted by the marketing department with modification d_1^S and concurrently to that her affiliation status is changed from undefined, indicating that she is a trainee, to D by the administration department with modification d_1^T. In Fig. 48, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the deletion of Lily Lee in the target domain. Conversely, in Fig. 49, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the change of Lily's status in the source domain, i.e., since in the source domain only personnel of the marketing department are administered, she is deleted due to her transfer to the development department. The modifications d_1^T and $d_{1,F}^T$ are in delete-use conflict, i.e., the node of Lily Lee is deleted by $d_{1,F}^T$ but is used by d_1^T to change the affiliation status. Furthermore, modifications d_1^S and $d_{1,B}^S$ are in delete-delete conflict.

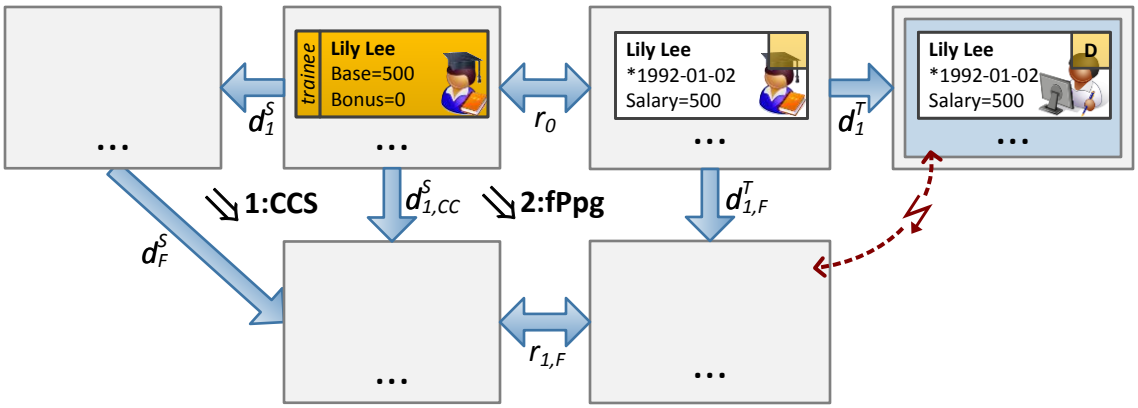


Figure 48: Transfer of Lily Lee - $fPpg$ first

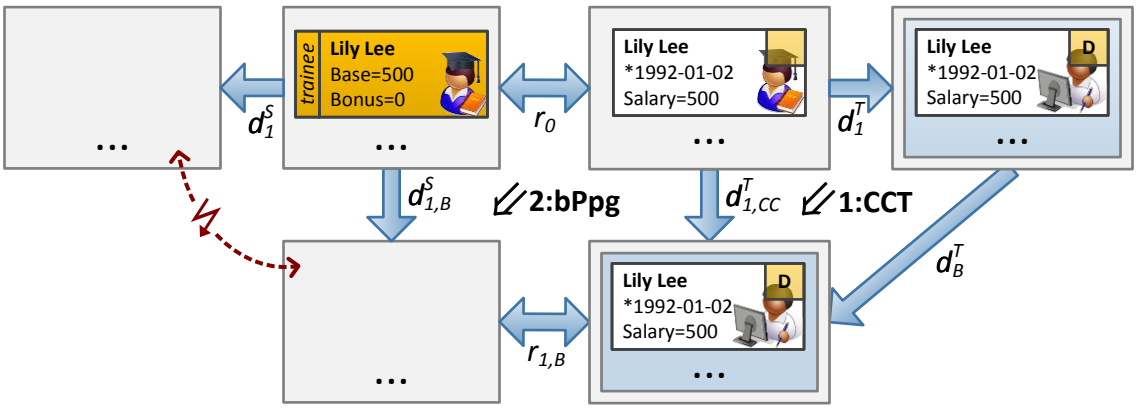


Figure 49: Transfer of Lily Lee - $bPpg$ first

Example 5.16 (Case 16 - Switch of Willy Wilson to a permanent position)**.** Willy Wilson as a trainee of the marketing department is switching to a permanent position in this department. Therefore, in Fig. 50 and Fig. 51, his trainee status is switched from true to false by the marketing department with modification d_1^S and concurrently to that his affiliation status is changed from *undefined*, indicating that he is a trainee, to M by the administration department with modification d_1^T. In Fig. 50, modification d_1^S with $d_1^S = d_{1,CC}^S$ is forward propagated to the administration department first, leading to modification $d_{1,F}^T$ which reflects the switching of the trainee status in the target domain, i.e. his affiliation status is changed from *undefined* to M. Conversely, in Fig. 51, modification d_1^T with $d_1^T = d_{1,CC}^T$ is backward propagated to the marketing department first, leading to modification $d_{1,B}^S$ which reflects the change of Willy's affiliation status in the source domain, i.e., his trainee status is changed from true to false. The modifications d_1^T and $d_{1,F}^T$ are conflict-free but modifications d_1^S and $d_{1,B}^S$ are in delete-delete conflict, i.e. concurrently changing the trainee status requires the concurrent deletion of its value first. Analogously to Ex. 5.14, this leads to the observation, that switching the order of forward and backward propagation steps may lead to different synchronisation results as illustrated by the complete synchronisation Ex. 6.16 in Sec. 6.

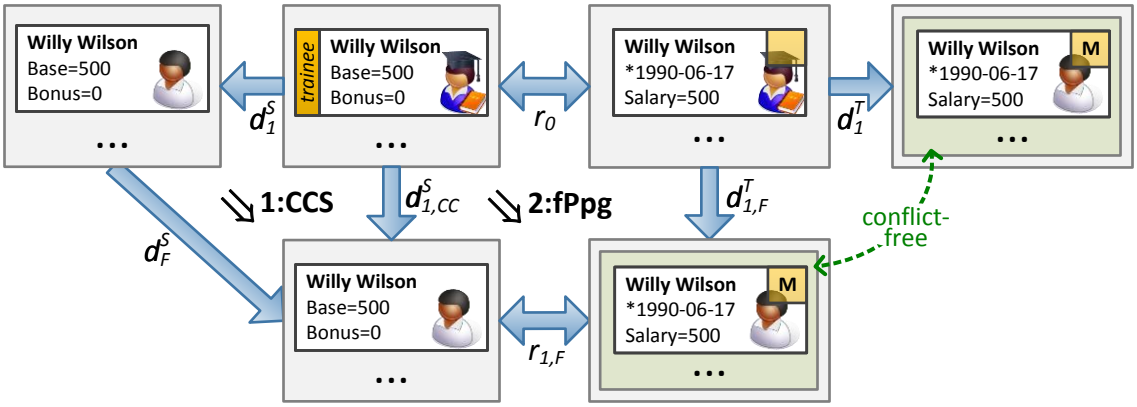


Figure 50: Switch of Willy Wilson to a permanent position - $fPpg$ first

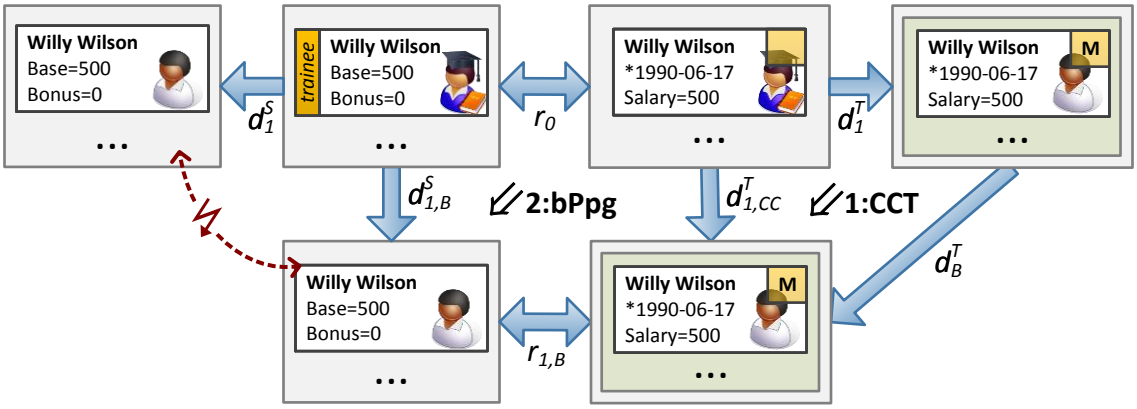


Figure 51: Switch of Willy Wilson to a permanent position - $bPpg$ first

Example 5.17 (Summary of all concurrent changes)**.** As mentioned in our introduction of the running example in Sec. 2, all cases we presented in detail in the current section are contained in one update in the source domain d_1^S, and one update in the target domain d_1^T, which are both performed concurrently. The following figures summarise all cases, where Fig. 52 illustrate the :CCS and :fPpg operations and Fig. 53 illustrate the :CCT and :bPpg operations, respectively.

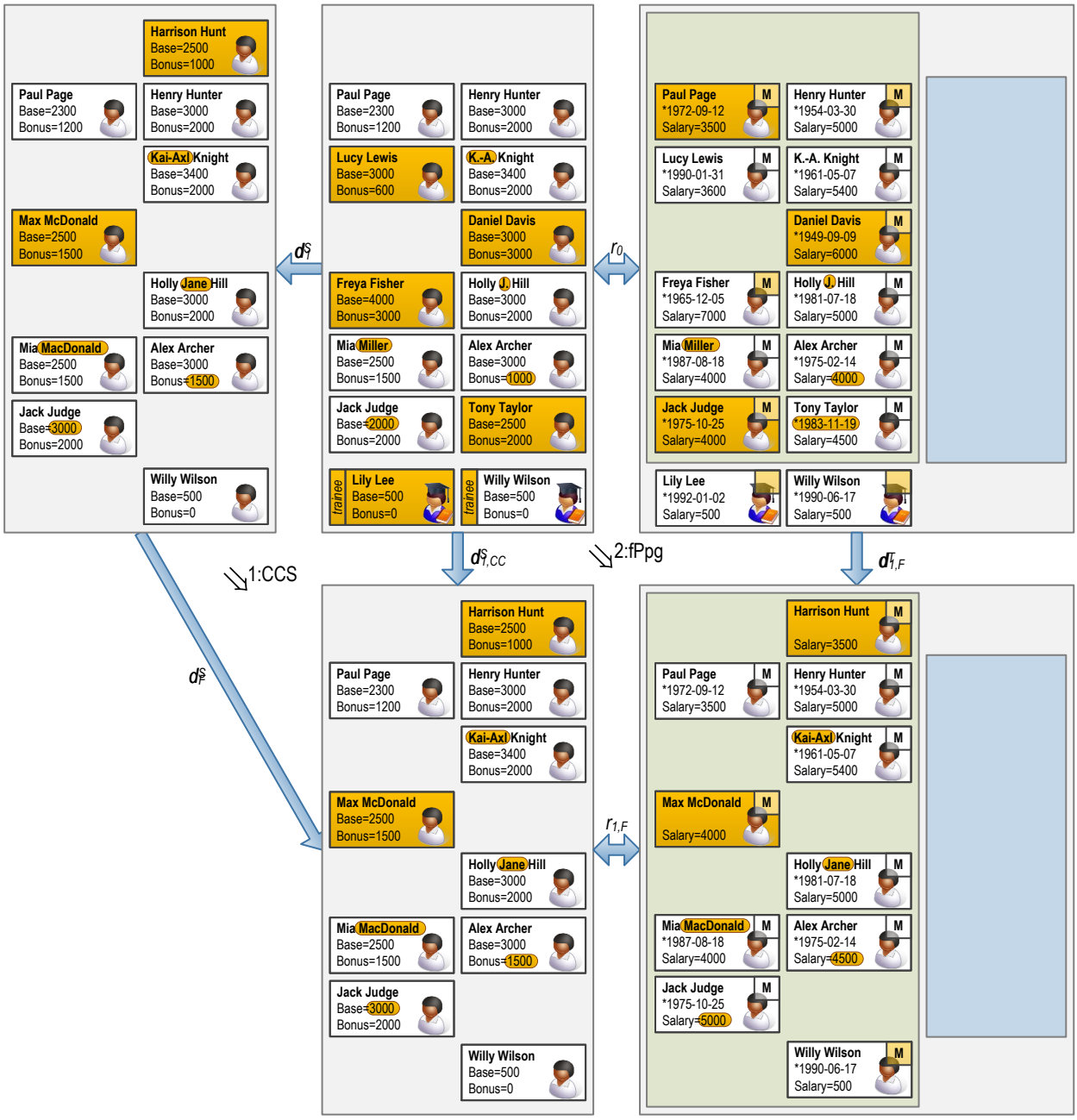


Figure 52: All cases CCS and fPpg operation

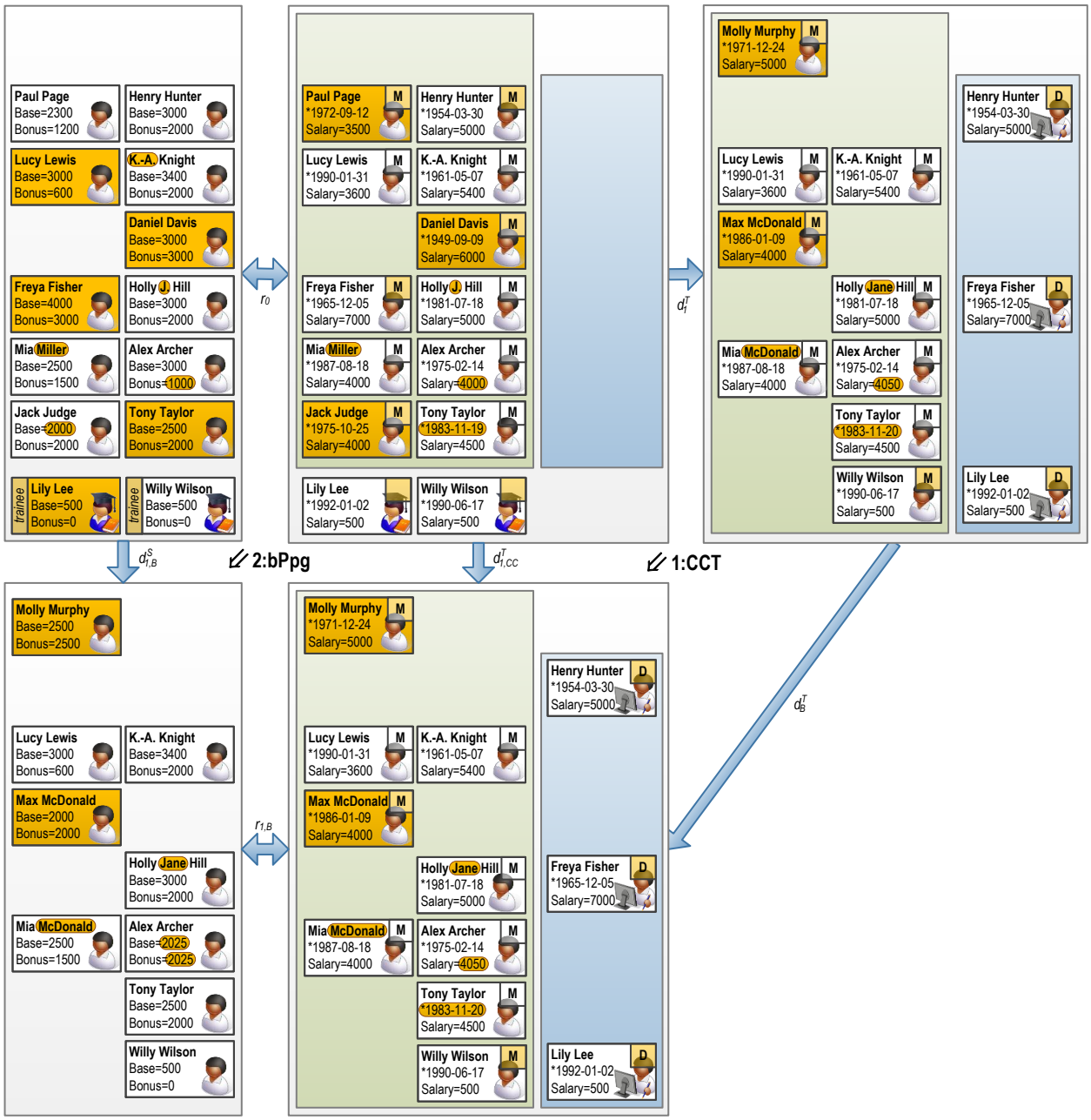


Figure 53: All cases CCT and bPpg operation

6 Conflict Resolution

Two concurrent source model updates (resp. target model updates) can cause conflicts. Two graph modifications m_1, m_2 with $m_i = (G \leftarrow D_i \rightarrow H_i), (i = 1,2)$ are conflict-free if they do not interfere with each other, i.e. each modification preserves those graph elements that are used by the other one to perform its changes, respectively. Otherwise both graph modifications are in conflict, i.e. one modification deletes a graph element that is used by the other one to perform its changes. Two types of conflicts can occur: (1) *delete-delete conflict*: the same graph element is deleted by m_1 and m_2, or (2) *delete-insert conflict*: m_1 deletes a node which is source or target of a newly inserted edge by m_2 (or vice versa for a deleting modification m_2). We now review the general merge construction with conflict resolution given in [6] where two graph modifications m_1 and m_2 are merged leading to a merged graph modification $(G \leftarrow \overline{D} \rightarrow H)$ denoted by m so that the changes of m_1 and m_2 are reflected appropriately by m, i.e. in case of conflicting modifications m_1 and m_2, insertion of graph elements takes priority over deletion of graph elements in m.

The main effects of the general merge construction for given non-conflicting modifications m_1, m_2 are summarized in the following Concept 2 and for given conflicting modifications m_1, m_2 where a conflict resolution is needed in Concept 3 (see also Thm. 2 and 3 in [6] for the properties of the resulting merged graph modification).

Concept 2 (General merge construction without conflict resolution)**.** Given two non-conflicting graph modifications m_1, m_2 with $m_i = (G \leftarrow D_i \rightarrow H_i), (i = 1,2)$. The general merge construction leads to a merged graph modification m with $m = (G \leftarrow \overline{D} \rightarrow H)$ having the following properties:

1. If m_1 and m_2 preserve graph element $x \in G$, then x is also preserved by m.
2. If m_1 creates graph element $x \in H_1$ or m_2 creates $x \in H_2$, then m creates $x \in H$.
3. If m_1 (resp. m_2) deletes graph element $x \in G$ that is preserved by m_2 (resp. m_1), then x is deleted by m. Note that x is not subject to an edge insertion by m_1 or m_2 as m_1 and m_2 are conflict-free.

Concept 3 (Tentative conflict resolution by general merge construction)**.** Given two conflicting graph modifications m_1, m_2 with $m_i = (G \leftarrow D_i \rightarrow H_i), (i = 1,2)$. The general merge construction leads to a merged graph modification m with $m = (G \leftarrow \overline{D} \rightarrow H)$ having the following properties:

1. Properties 1 and 2 of Concept 2 are valid.
2. If m_1 and m_2 delete the same graph element $x \in G$, i.e. (m_1, m_2) are in delete-delete conflict, then x is deleted by m.
3. If there is an edge e created by m_2 and a node x with $x = s(e)$ or $x = t(e)$ being preserved by m_2, but deleted by m_1, i.e. (m_1, m_2) are in delete-insert conflict, then x is preserved and e is created by m whereas all other dangling edges of x are deleted (and vice versa for (m_2, m_1) being in delete-insert conflict).

The following examples 6.1 to 6.16 complete the examples 5.1 to 5.16 of Sec. 5 in view of synchronising models after they were concurrently modified. Therefore, conflicts between conflicting concurrent model modifications are resolved by means of the general merge construction and the resulting merged modifications are forward (resp. backward) propagated to the target (resp. source) domain. All examples are based on the application scenario of Sec. 2 where models of a company's marketing and administration department are modified concurrently by a model modification in each domain, respectively. The marketing department is denoted the source domain and the administration department the target domain of the model synchronisation.

Ex. 6.14 and Ex. 6.16 unfolds an interesting property of concurrent model synchronisations, that changing the order of the forward and backward propagation steps may lead to different conflict and synchronisation results.

Example 6.1 (Case 1 - Appointment of Molly Murphy)**.** Molly Murphy as a new member of the marketing department is added to the marketing department by the administration department with modification d_1^T but not by the marketing department itself indicated by the identity modification d_1^S in Fig. 54 and Fig. 55.

After forward propagating modification d_1^S to the target domain first in Fig. 54, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ preserves the addition of Molly Murphy without performing any other changes, i.e., furthermore $d_{2,CC}^T$ reflects the non-changing nature of d_1^S in the target domain. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects the addition of Molly Murphy in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 55, the general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,BC}$ with $d'^S_{2,BC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ reflects the addition of Molly Murphy in the source domain without performing any other changes, i.e., furthermore $d_{2,CC}^S$ preserves the non-changing nature of d_1^S. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which reflects the non-changing nature of d_1^S in the target domain..

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

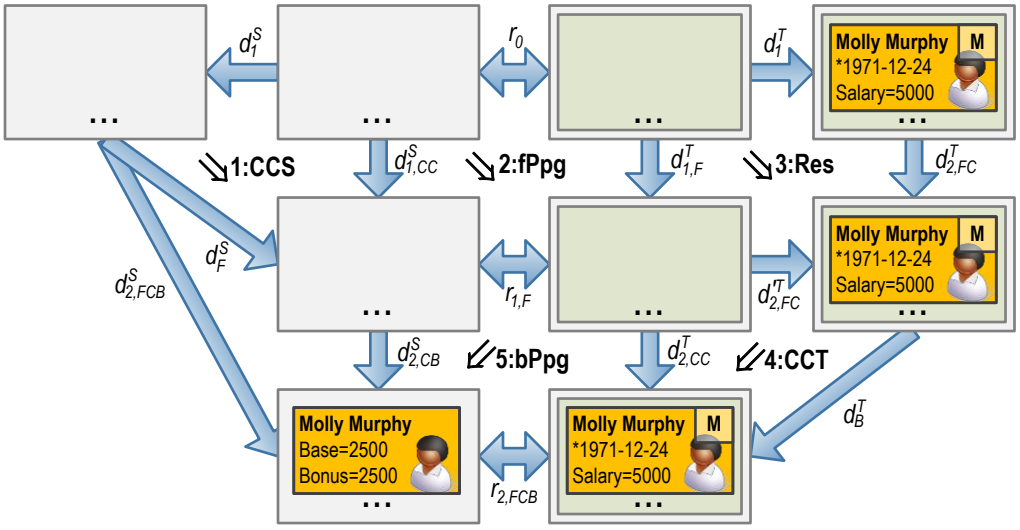


Figure 54: Appointment of Molly Murphy - fSync

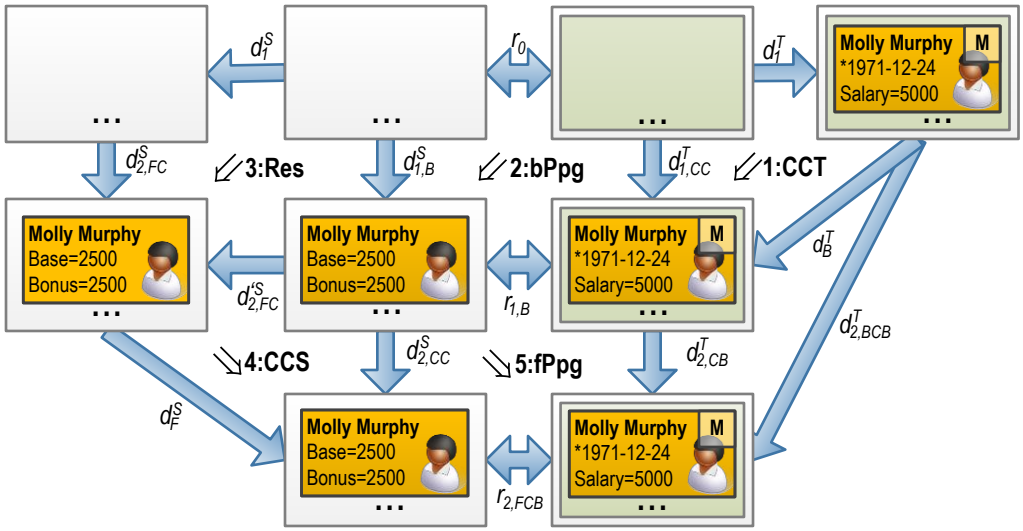


Figure 55: Appointment of Molly Murphy - bSync

Example 6.2 (Case 2 - Appointment of Harrison Hunt)**.** Harrison Hunt as a new member of the marketing department is added by the marketing department with modification d_1^S but not by the administration department indicated by the identity modification d_1^T in Fig. 56 and Fig. 57.

After forward propagating modification d_1^S to the target domain first in Fig. 56, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ reflects the addition of Harrison Hunt in the target domain without performing any other changes, i.e., $d_{2,CC}^T$ preserves the non-changing nature of d_1^T. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects the non-changing nature of d_1^T in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 57, the general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ preserves the addition of Molly Murphy without performing any other changes, i.e., $d_{2,CC}^S$ reflects the non-changing nature of d_1^T in the source domain. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which reflects the addition of Harrison Hunt in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

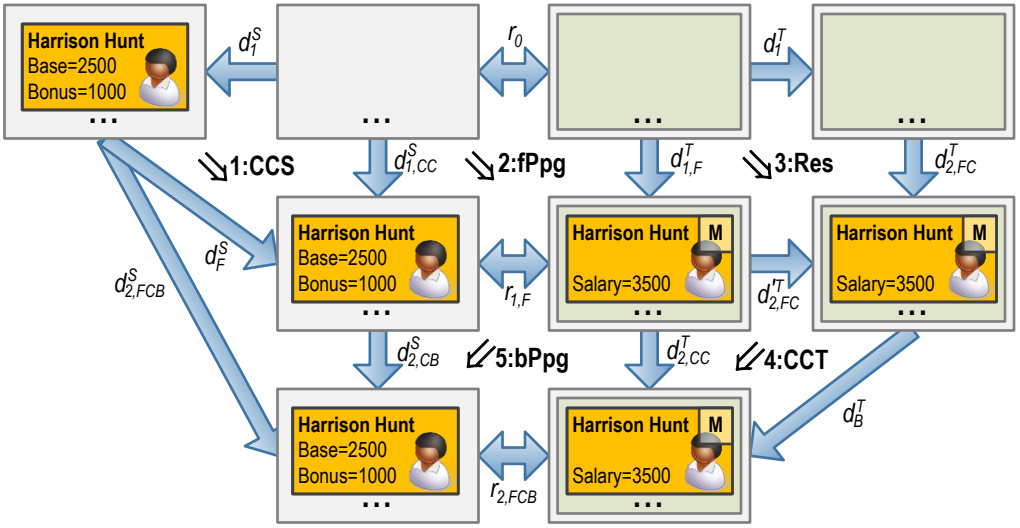


Figure 56: Appointment of Harrison Hunt - fSync

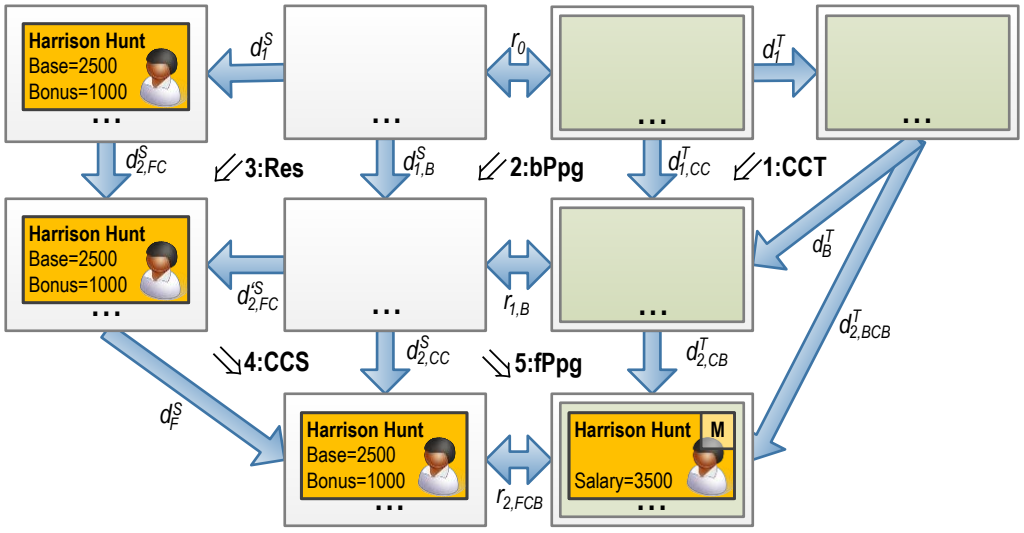


Figure 57: Appointment of Harrison Hunt - bSync

Example 6.3 (Case 3 - Resignation of Paul Page)**.** Paul Page as a former member of the marketing department is removed by the administration department with modification d_1^T due to his resignation but not by the marketing department itself indicated by the identity modification d_1^S in Fig. 58 and Fig. 59.

After forward propagating modification d_1^S to the target domain first in Fig. 58, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ preserves the deletion of Paul Page without performing any other changes, i.e., furthermore $d_{2,CC}^T$ reflects the non-changing nature of d_1^S in the target domain. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects the deletion of Paul Page in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 59, the

general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ reflects the deletion of Paul Page in the source domain without performing any other changes, i.e., furthermore $d_{2,CC}^S$ preserves the non-changing nature of d_1^S. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which reflects the non-changing nature of d_1^S in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

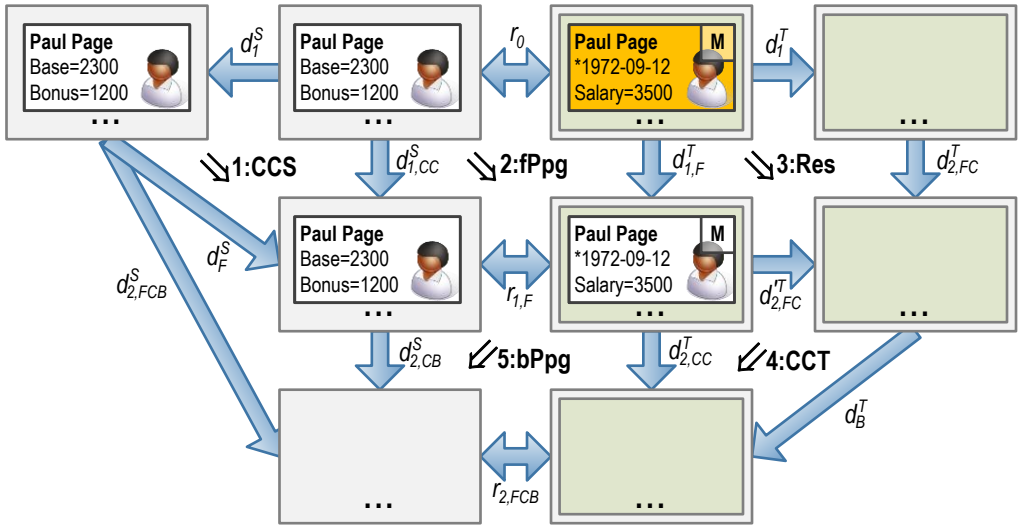


Figure 58: Resignation of Paul Page - fSync

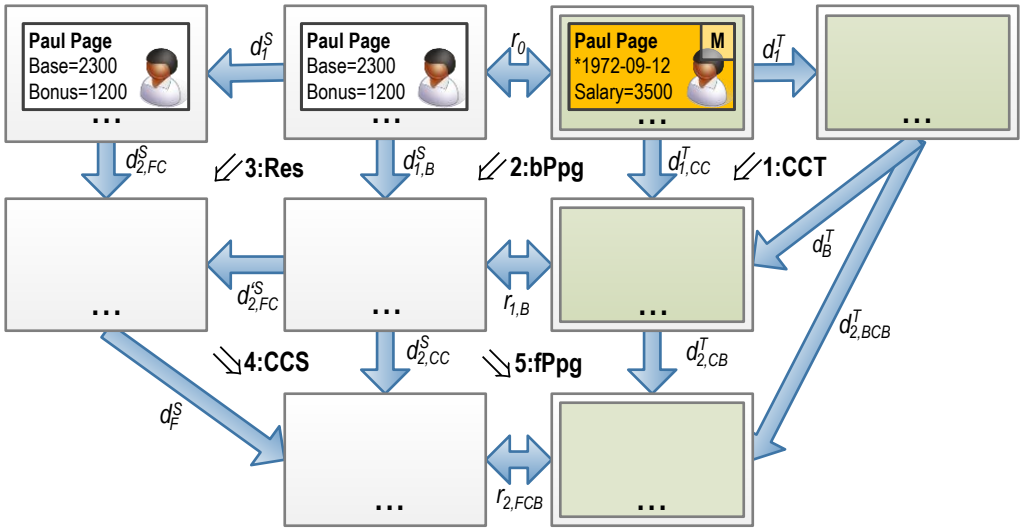


Figure 59: Resignation of Paul Page - bSync

Example 6.4 (Case 4 - Transfer of Henry Hunter)**.** Henry Hunter as a former member of the marketing department is transferred to the development department. Therefore, his affiliation status is changed from M to D by the administration department with modification d_1^T whereas no changes are performed by the marketing department itself indicated by the identity modification d_1^S in Fig. 60 and Fig. 61.

After forward propagating modification d_1^S to the target domain first in Fig. 60, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ reflects the status change without performing any other changes, i.e., furthermore $d_{2,CC}^T$ reflects the non-changing nature of d_1^S in the target domain. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects the affiliation status change of Henry Hunter in the source domain, i.e., since only personnel of the marketing department are administered in the source domain, Henry Hunter is deleted due to his transfer to the development department.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 61, the general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ reflects the affiliation status change of Henry Hunter in the source domain by deleting his node without performing any other changes, i.e., furthermore $d_{2,CC}^S$ preserves the non-changing nature of d_1^S. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which reflects the non-changing nature of d_1^S in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

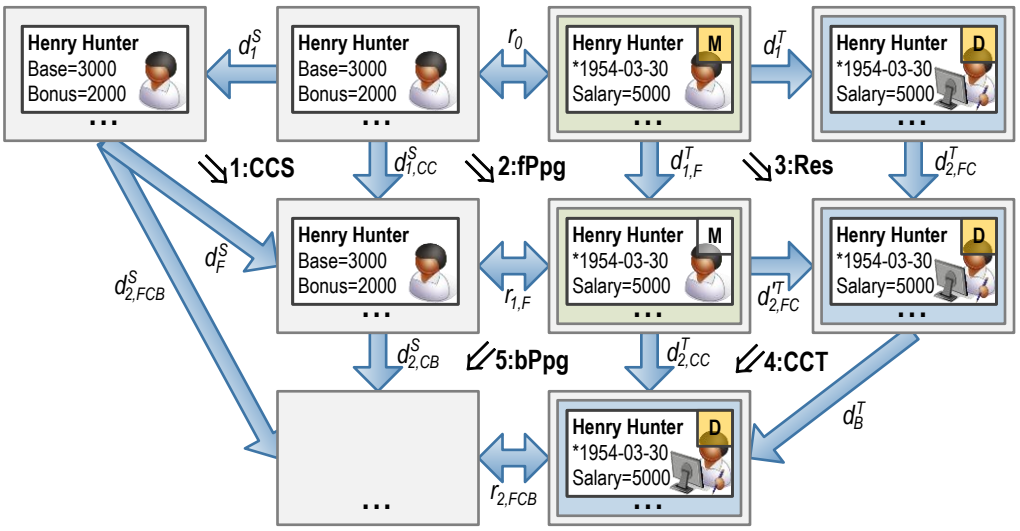


Figure 60: Transfer of Henry Hunter - fSync

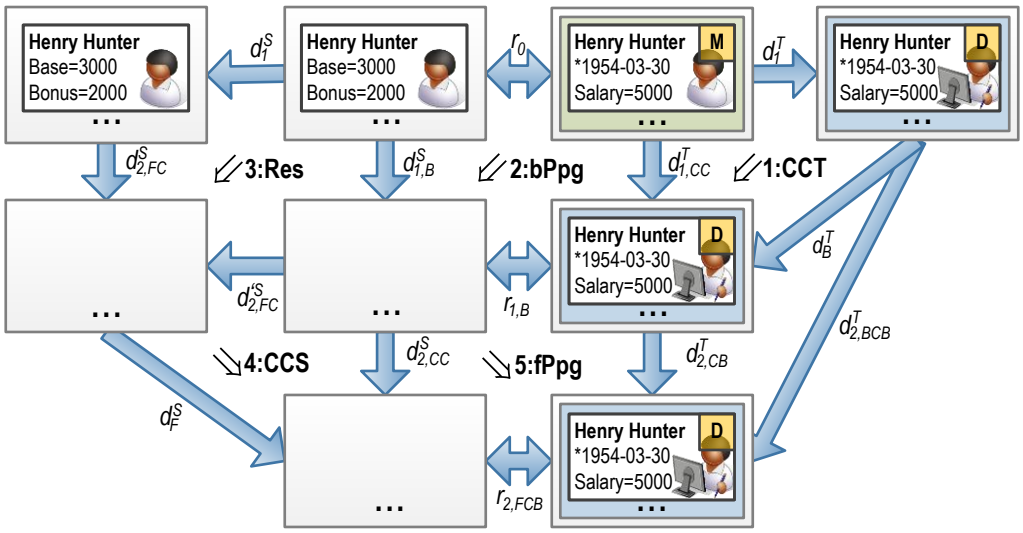


Figure 61: Transfer of Henry Hunter - bSync

Example 6.5 (Case 5 - Resignation Lucy Lewis)**.** Lucy Lewis as a former member of the marketing department resigns. Therefore, she is deleted by the marketing department with modification d_1^S but not by the administration department indicated by the identity modification d_1^T in Fig. 62 and Fig. 63.

After forward propagating modification d_1^S to the target domain first in Fig. 62, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ reflects the deletion of Lucy Lewis in the target domain without performing any other changes, i.e., furthermore $d_{2,CC}^T$ reflects the non-changing nature of d_1^T. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects the non-changing nature of d_1^T in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 63, the general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ preserves the deletion of Lucy Lewis without performing any other changes, i.e., furthermore $d_{2,CC}^S$ reflects the non-changing nature of d_1^T in the source domain. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which reflects the deletion of Lucy Lewis in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

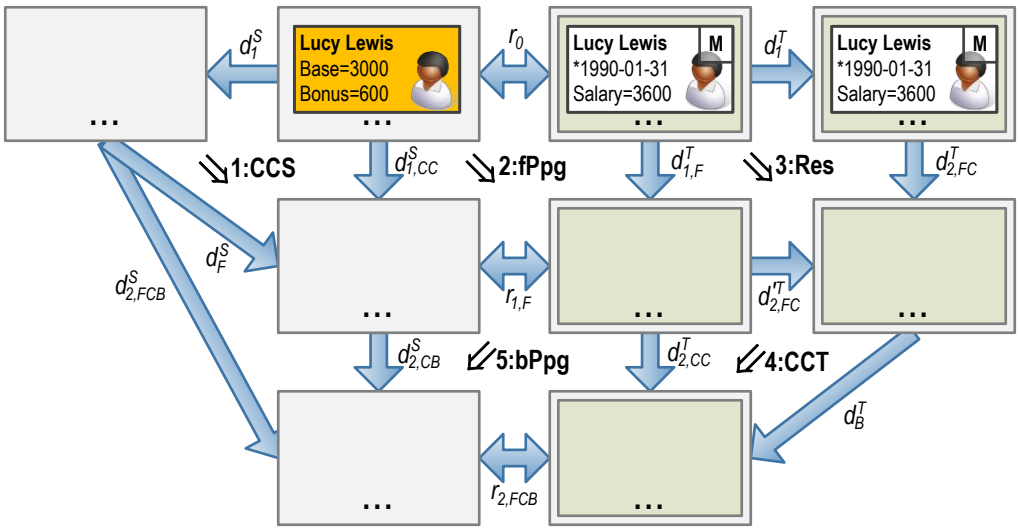


Figure 62: Resignation of Lucy Lewis - fSync

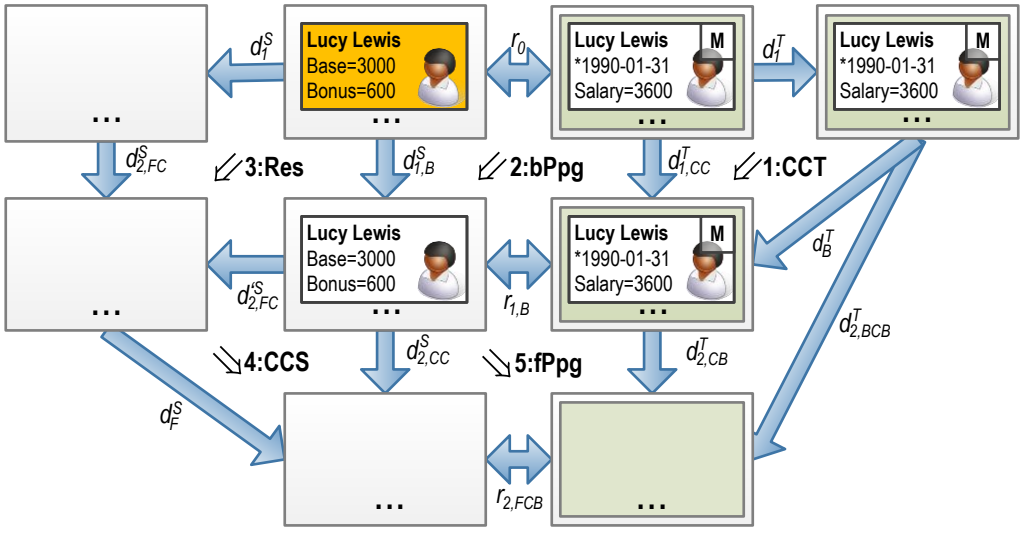


Figure 63: Resignation of Lucy Lewis - bSync

Example 6.6 (Case 6 - Renaming of K.-A. Knight)**.** The marketing department detects that the first name of *Kai-Axl Knight* is abbreviated in the model. Therefore, his first name is changed by the marketing department with modification d_1^S but not by the administration department indicated by the identity modification d_1^T in Fig. 64 and Fig. 65.

After forward propagating modification d_1^S to the target domain first in Fig. 64, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ reflects the change of the first name in the target domain without performing any other changes, i.e., furthermore $d_{2,CC}^T$ reflects the non-changing nature of d_1^T. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects the non-changing nature of d_1^T in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 65, the general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ preserves the change of the first name without performing any other changes, i.e., furthermore $d_{2,CC}^S$ reflects the non-changing nature of d_1^T in the source domain. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which reflects the change of the first name in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \xleftrightarrow{r_{2,FCB}} G_{2,FCB}^T$ and $G_{2,BCF}^S \xleftrightarrow{r_{2,BCF}} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

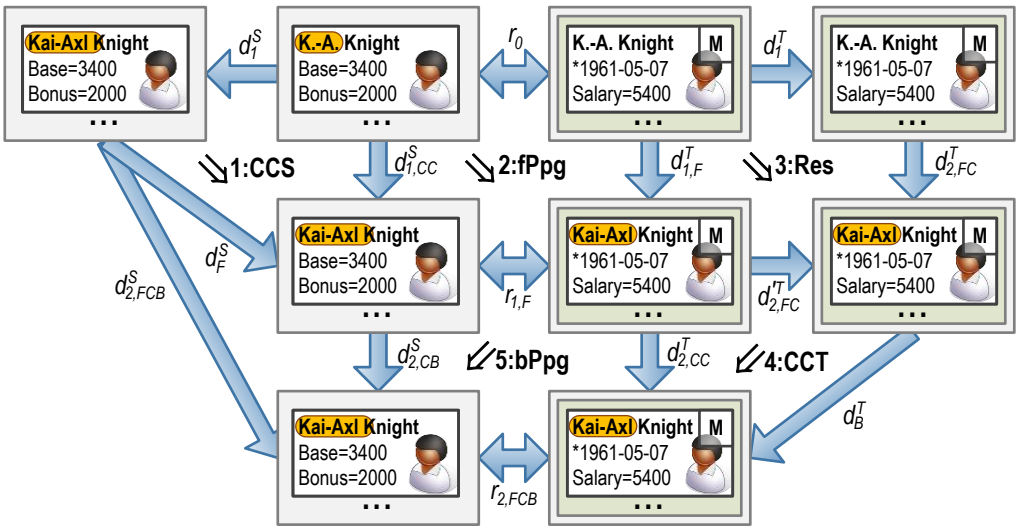


Figure 64: Renaming of K.-A. Knight - fSync

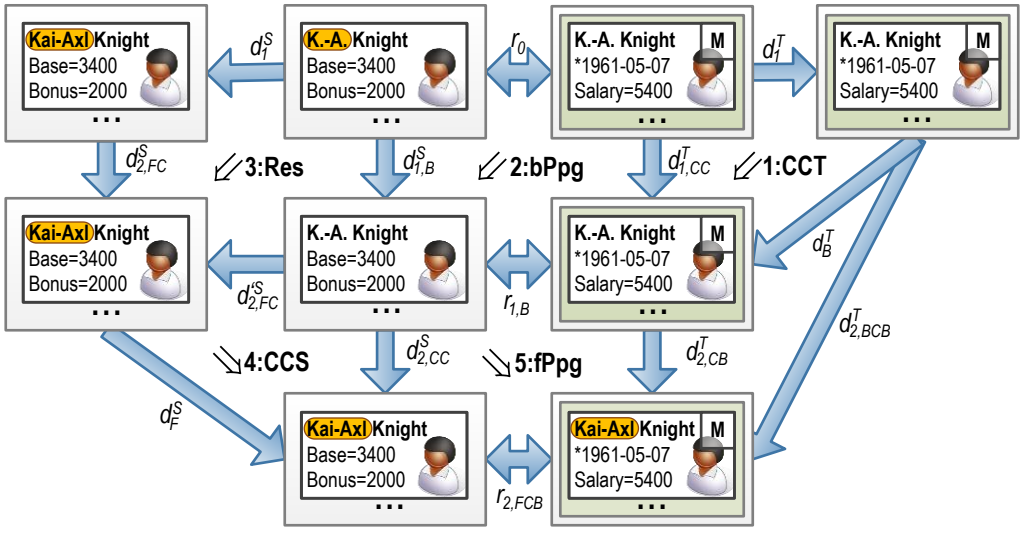


Figure 65: Renaming of K.-A. Knight - bSync

Example 6.7 (Case 7 - Appointment of Max McDonald)**.** Max McDonald is appointed as a new member of the marketing department. Therefore, he is concurrently added by the marketing and administration department with modifications d_1^S and d_1^T in Fig. 66 and Fig. 67.

After forward propagating modification d_1^S to the target domain first in Fig. 66, the general merge construction of the non-conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ reflects the addition of Max McDonald performed by modification d_1^S as well as the addition performed by modification d_1^T in the target domain. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which reflects both additions in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 67, the general merge construction of the non-conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ reflects both additions of Max McDonald in the source domain. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which also reflects both additions in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

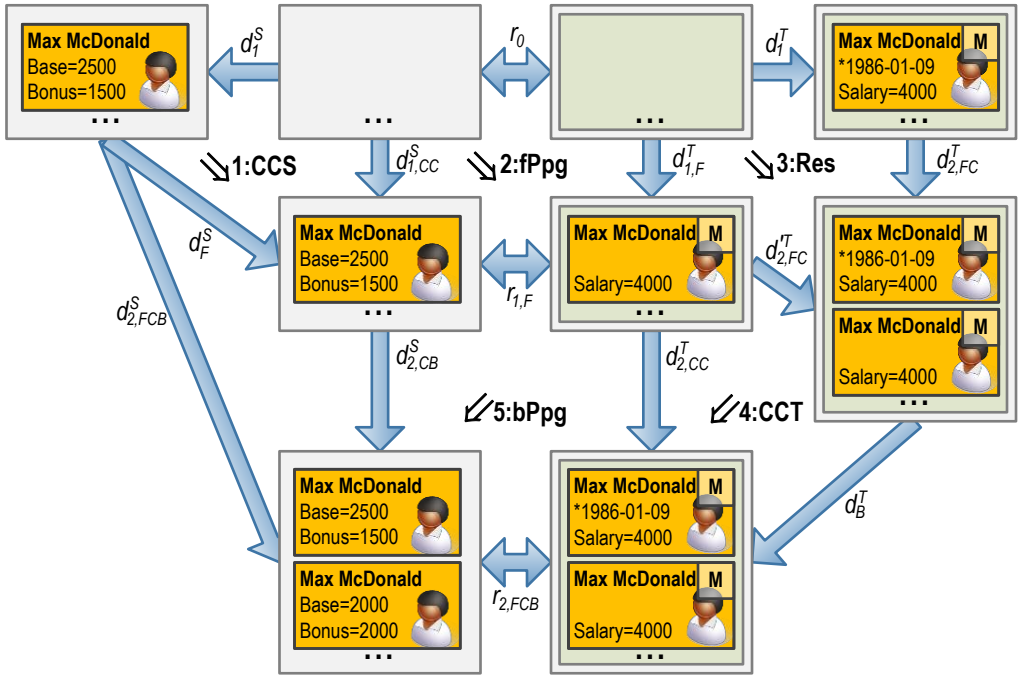


Figure 66: Appointment of Max McDonald - fSync

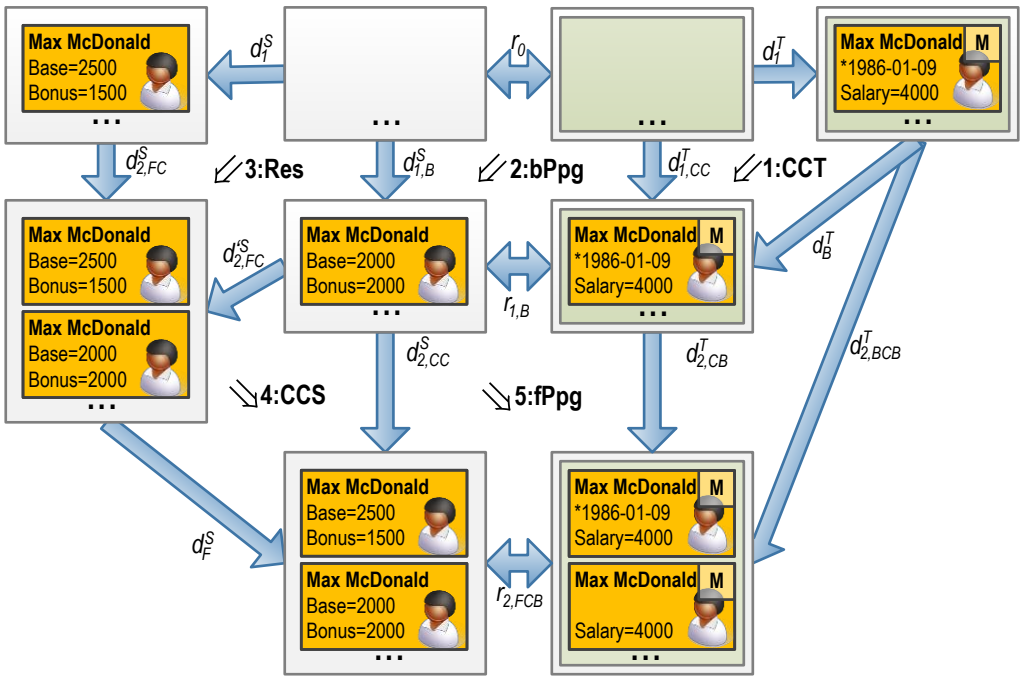


Figure 67: Appointment of Max McDonald - bSync

Example 6.8 (Case 8 - Resignation of Daniel Davis)**.** Daniel Davis as a former member of the marketing department resigns. Therefore, he is concurrently deleted by the marketing and administration department with modifications d_1^S and d_1^T in Fig. 68 and Fig. 69.

After forward propagating modification d_1^S to the target domain first in Fig. 68, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification $d_{2,CC}^T$ reflects the deletion of Daniel Davis in the target domain. Then, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ which also reflects the deletion in the source domain.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 69, the general merge construction of the conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d_{2,CC}^S$ so that modification $d_{2,CC}^S$ reflects the deletion of Daniel Davis in the source domain. Then, modification $d_{2,CC}^S$ is forward propagated to the target domain leading to modification $d_{2,CB}^T$ which also reflects the deletion in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G_{2,FCB}^S \overset{r_{2,FCB}}{\longleftrightarrow} G_{2,FCB}^T$ and $G_{2,BCF}^S \overset{r_{2,BCF}}{\longleftrightarrow} G_{2,BCF}^T$ with $G_{2,FCB}^S = G_{2,BCF}^S, G_{2,FCB}^T = G_{2,BCF}^T$ and $r_{2,FCB} = r_{2,BCF}$.

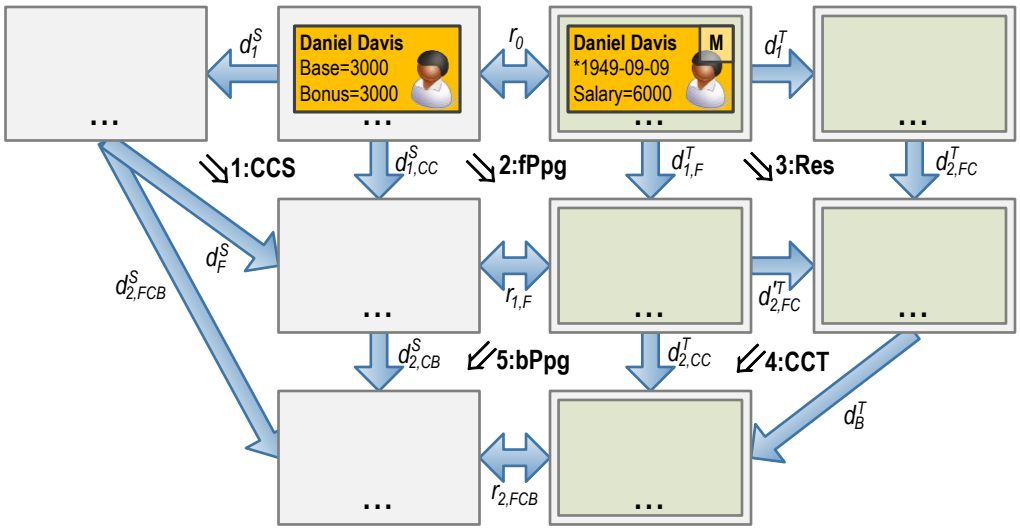


Figure 68: Resignation of Daniel Davis - fSync

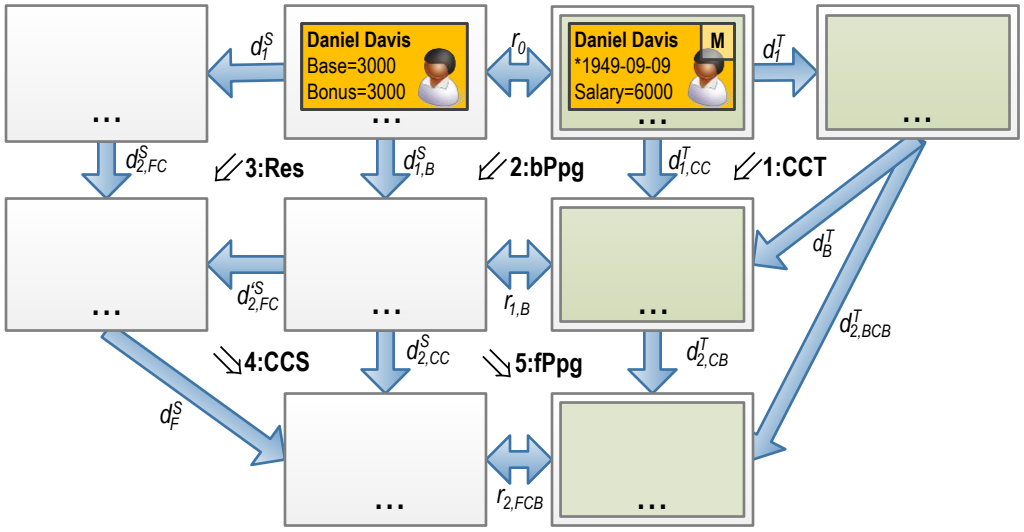


Figure 69: Resignation of Daniel Davis - bSync

Example 6.9 (Case 9 - Transfer of Freya Fisher)**.** Freya Fisher as a former member of the marketing department is transferred to the development department. Therefore, she is deleted by the marketing with modification d_1^S and concurrently to that, her affiliation status is changed from M to D by the administration department with modification d_1^T in Fig. 70 and Fig. 72.

After forward propagating modification d_1^S to the target domain first in Fig. 70, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ preserves the affiliation status change of Freya Fisher but neglects the deletion of Freya Fisher in the target domain due to the higher priorisation of inserting graph elements over deletion of graph elements in the conflict resolution construction. However, it is left to the user to choose between the deletion of Freya Fisher and

the status change, i.e., the insertion of a new affiliation status with value D while deleting the old status of value M. Details of the manual conflict resolution step (3:Res) of the fSync operation are illustrated in Fig. 71. In the following, the status change is chosen to be considered. Hence, modification $d^T_{2,CC}$ is backward propagated to the source domain leading to modification $d^S_{2,CB}$ with modification sequence $G^S_0 \overset{d^S_{1,CC}}{\Longrightarrow} G^S_{1,C} \overset{d^S_{2,CB}}{\Longrightarrow} G^S_{2,FCB}$ preserving the deletion of Freya Fisher in the source domain.

Conversely, after backward propagating modification d^T_1 to the source domain first in Fig. 72, the general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ preserves the deletion of Freya Fisher in the source domain. Then, modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ preserving the status change in the target domain.

Synchronisations fSync and bSync lead to the same synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$.

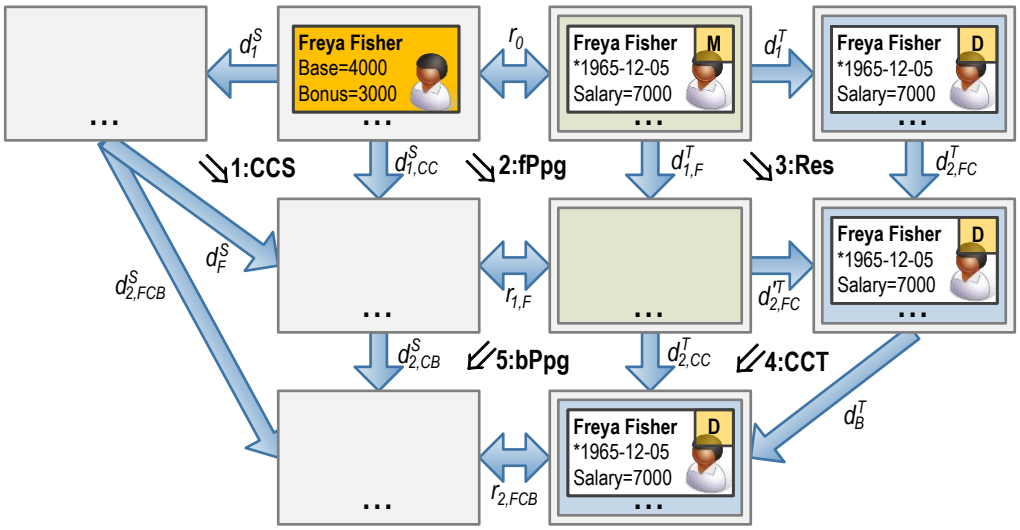


Figure 70: Transfer of Freya Fisher - fSync

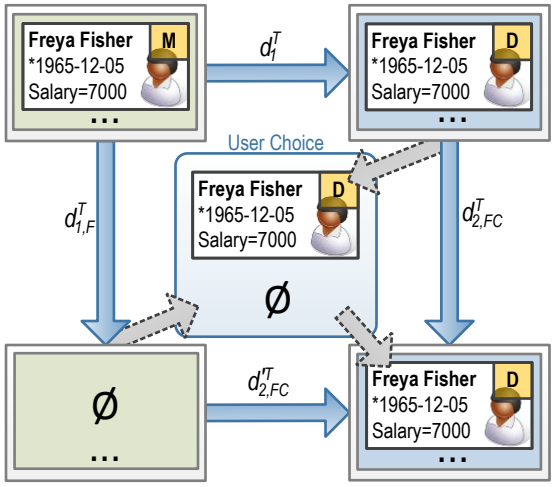


Figure 71: Case 9 - Step 3:Res in fSync operation

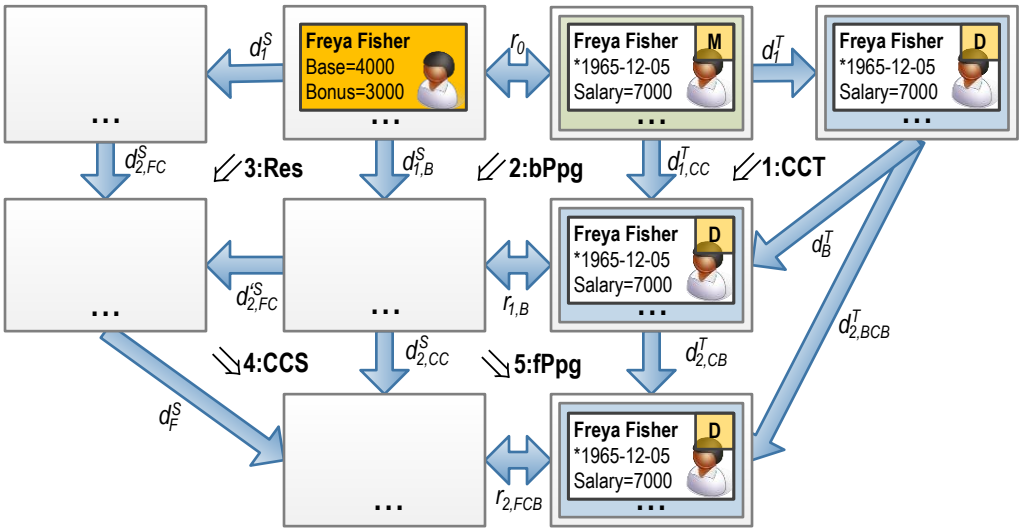


Figure 72: Transfer of Freya Fisher - bSync

Example 6.10 (Case 10 - Simultaneously rename Holly J. Hill)**.** The full first name of Holly Jane Hill is concurrently added by modifications d_1^S and d_1^T (see Fig. 73 and Fig. 75).

After forward propagating modification d_1^S to the target domain first in Fig. 73, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the addition of the full first name of Holly Jane Hill. However, modifications d_1^T and $d_{1,F}^T$ are in conflict due to the concurrent modification of the attribute FirstName. Thus, the user has to solve the conflict manually in selecting the correct first name, even if the change is performed on both sides correctly, so that the same first name is suggested twice. For details of the manual conflict resolution step (3:Res) of the fSync operation see the left-hand side of Fig. 74. After selecting one first name, modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ with modification sequence

$G_0^S \overset{d^S_{1,CC}}{\Longrightarrow} G_{1,C}^S \overset{d^S_{2,CB}}{\Longrightarrow} G_{2,FCB}^S$ preserving the change of Holly Jane Hill's first name.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 75, the general merge construction of the conflicting modifications $d_1^S, d_{1,B}^S$ leads to an already consistent graph $G_{2,BC}^S$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G_0^S \overset{d^S_{1,B}}{\Longrightarrow} G_{1,B}^S \overset{d^S_{2,CC}}{\Longrightarrow} G_{2,BCF}^S$ reflects the change of Holly Jane Hill's first name. Again, this leads to a attribute modification vs. attribute modification conflict, so that manual conflict resolution is necessary (similar to the fSync operation). For details see the right-hand side of Fig. 74. After selecting a first name, modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G_0^T \overset{d^T_{1,CC}}{\Longrightarrow} G_{1,C}^T \overset{d^T_{2,CF}}{\Longrightarrow} G_{2,BCF}^T$ preserving the change of Holly Jane Hill's first name.

Synchronisations fSync and bSync lead to the same synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$.

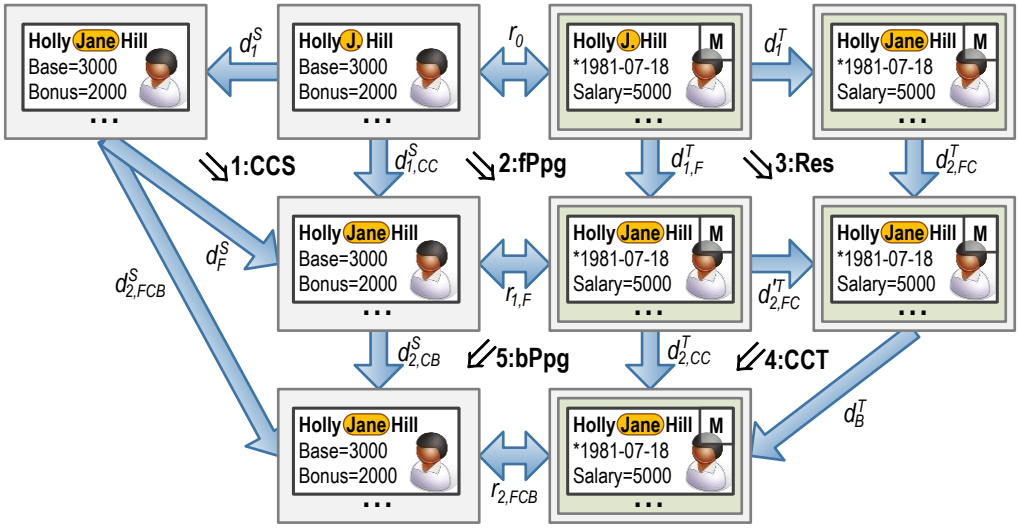


Figure 73: Case 10 - fSync operation

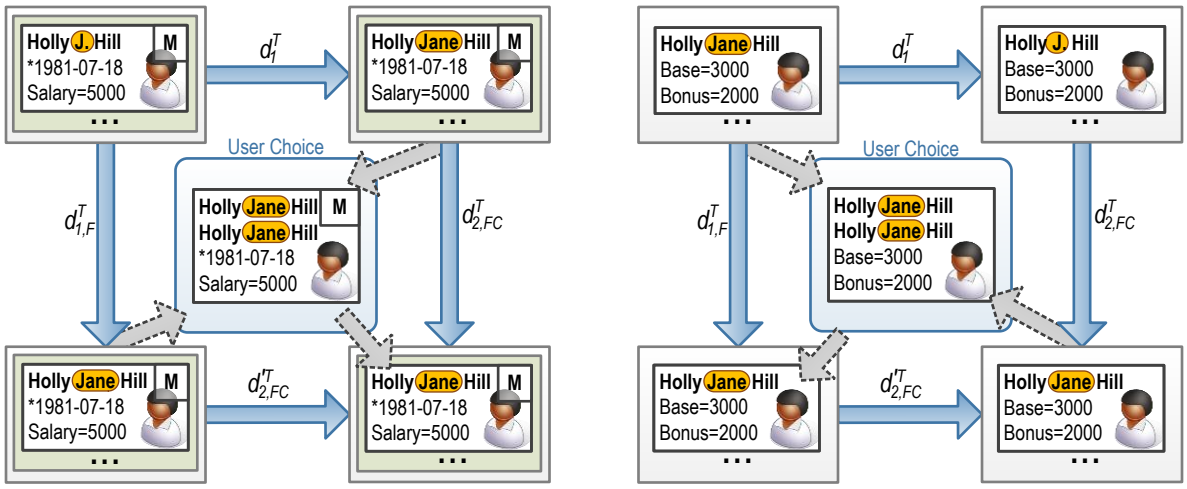


Figure 74: Case 10 - Step 3:Res in fSync operation on the left and in bSync operation on the right

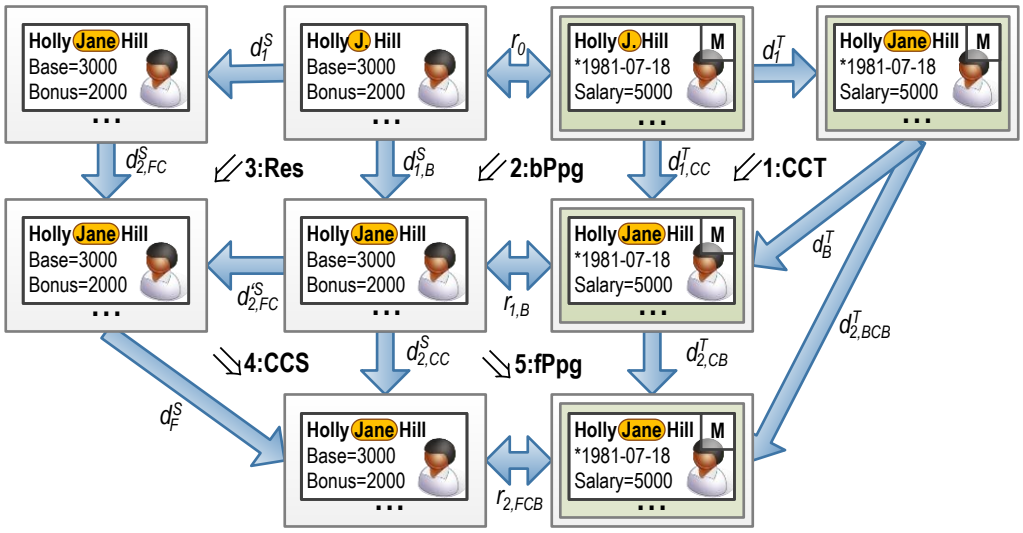


Figure 75: Case 10 - bSync operation

Example 6.11 (Case 11 - Simultaneously rename Mia Miller)**.** Due to marriage, the name of Mia Miller changes to McDonald. This change is concurrently added by modifications d_1^S and d_1^T (see Fig. 76 and Fig. 78), but a typing error is introduced in the source domain reflected by modification d_1^S.

After forward propagating modification d_1^S to the target domain first in Fig. 76, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the change of the last name of Mia Miller to Mia MacDonald. However, modifications d_1^T and $d_{1,F}^T$ are in conflict due to the concurrent modification of the attribute LastName. Thus, the user has to solve the conflict manually in selecting the correct name, which is McDonald. Both possibilities for the last name are suggested by the algorithm. Details of the manual conflict resolution step (3:Res) of the fSync operation are illustrated on the left-hand side of Fig. 77. After selecting the correct last

name, modification $d^T_{2,CC}$ is backward propagated to the source domain leading to modification $d^S_{2,CB}$ with modification sequence $G^S_0 \overset{d^S_{1,CC}}{\Longrightarrow} G^S_{1,C} \overset{d^S_{2,CB}}{\Longrightarrow} G^S_{2,FCB}$ preserving the change of the last name.

Conversely, after backward propagating modification d^T_1 to the source domain first in Fig. 78, the general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ reflects the change of Mia Miller's last name to McDonald. Again, this leads to a attribute modification vs. attribute modification conflict, so that manual conflict resolution is necessary (similar to the fSync operation). For details see the right-hand side of Fig. 77. After selecting the correct last name, modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ preserving the change of the last name.

Synchronisations fSync and bSync lead to the same synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$, but only if the manual conflict resolution is performed so that the correct name is selected.

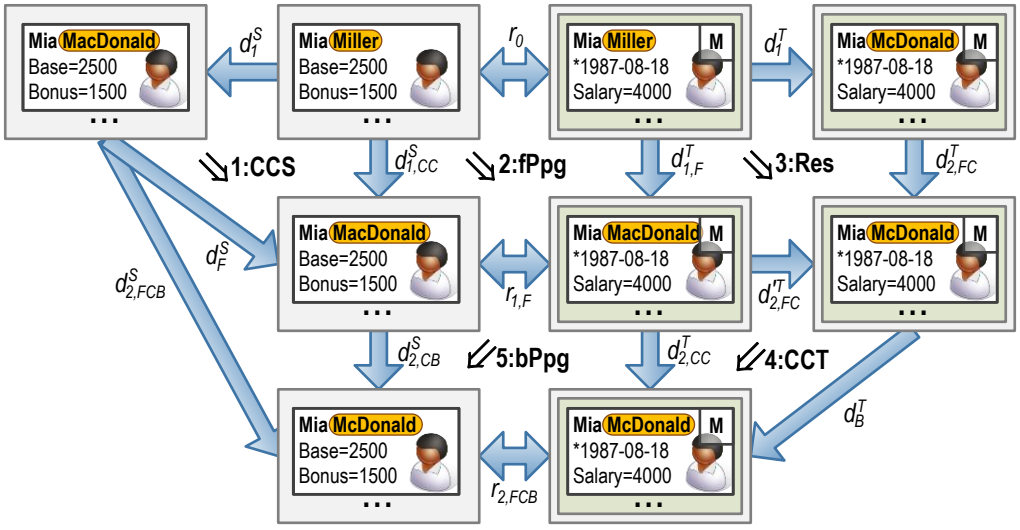


Figure 76: Case 11 - fSync operation

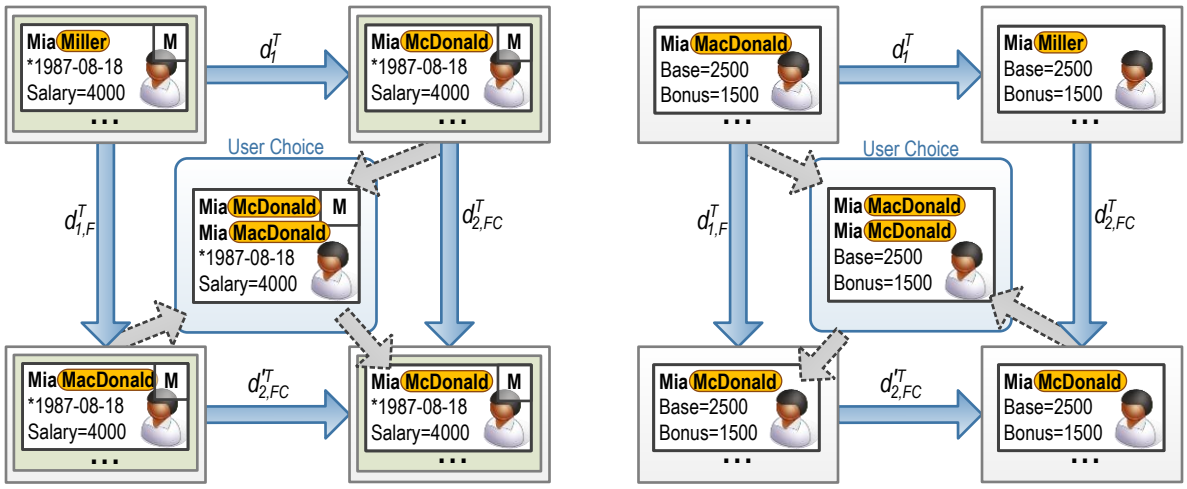


Figure 77: Case 11 - Step 3:Res in fSync operation on the left and in bSync operation on the right

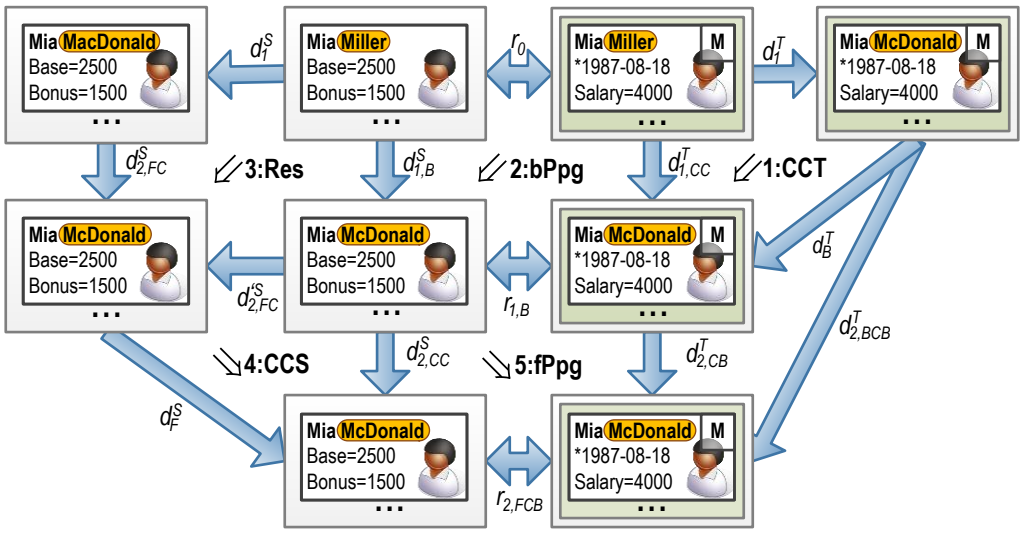


Figure 78: Case 11 - bSync operation

Example 6.12 (Case 12 - Simultaneously change salary and bonus of Alex Archer)**.** In the source domain, the Bonus salary of Alex Archer gets increased due to performing a good job, which is reflected by model update d_1^S (see Fig. 79). Concurrently, in the target model, his Salary get increased, in order to compensate the inflationary adjustment. This change is reflected by modification d_1^T (see Fig. 81).

After forward propagating modification d_1^S to the target domain first in Fig. 79, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the change of the Bonus salary of Alex Archer.. However, modifications d_1^T and $d_{1,F}^T$ are in conflict due to the concurrent modification of the attribute Salary. Thus, the user has to solve the conflict manually in selecting the correct value for the Salary. The algorithm suggest two possibilities: Salary = 4050 and Salary = 4500. Both modifications are performed due to different reasons, so that during manual conflict

resolution, the user enters the correct value manually: Salary = 4550. The manual conflict resolution step (3:Res) of the fSync operation is illustrated on the left-hand side of Fig. 80. After entering the correct value, modification $d^T_{2,CC}$ is backward propagated to the source domain leading to modification $d^S_{2,CB}$ with modification sequence $G^S_0 \overset{d^S_{1,CC}}{\Longrightarrow} G^S_{1,C} \overset{d^S_{2,CB}}{\Longrightarrow} G^S_{2,FCB}$. The backward propagation is not able to derive the correct values for the attributes Base and Bonus. Therefore, as already explained in Ex. 5.12, the values are calculated as follows: $Salary = 2 \cdot Base = 2 \cdot Bonus$.

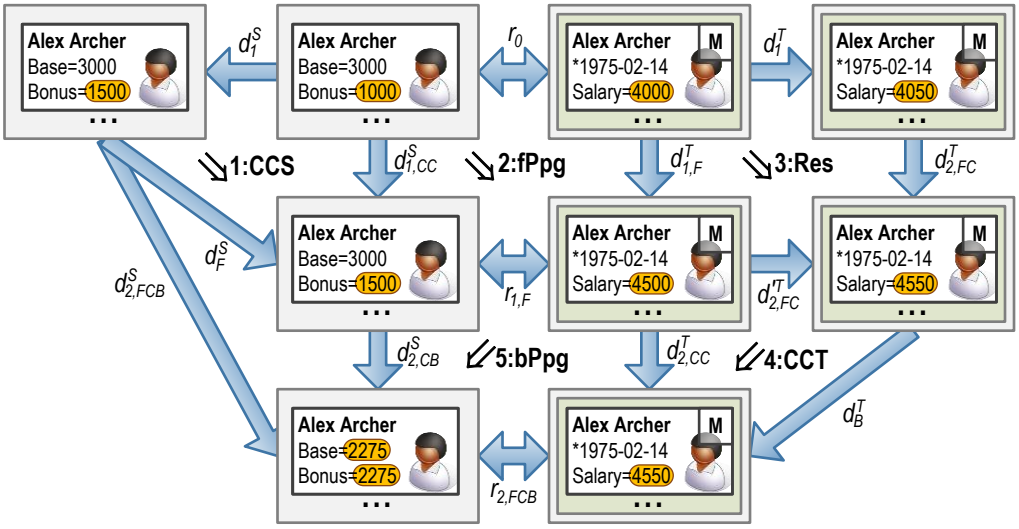


Figure 79: Case 12 - fSync operation

Conversely, after backward propagating modification d^T_1 to the source domain first in Fig. 81, the general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ reflects the change of Alex Archer's Salary. Again, this leads to a attribute modification vs. attribute modification conflict, so that manual conflict resolution is necessary (similar to the fSync operation), where the user enters the correct values for Base and Bonus manually, without choosing one of the proposed values. For details see the right-hand side of Fig. 80. After entering the correct values, modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ reflecting the change of the Salary.

Synchronisations fSync and bSync lead to different synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$, because the algorithm is not able to propagate the changes correctly, i.e., in this case, the addition of both changes will be correct, whereas the compensation of the inflationary adjustment will be added to the Base. These kind of information is not reflected by any update and will always need manual intervention. Furthermore, the bSync operation will result in the desired model, the fSync operation will not. Consequently, the prioritisation of the bPpg operation is necessary.

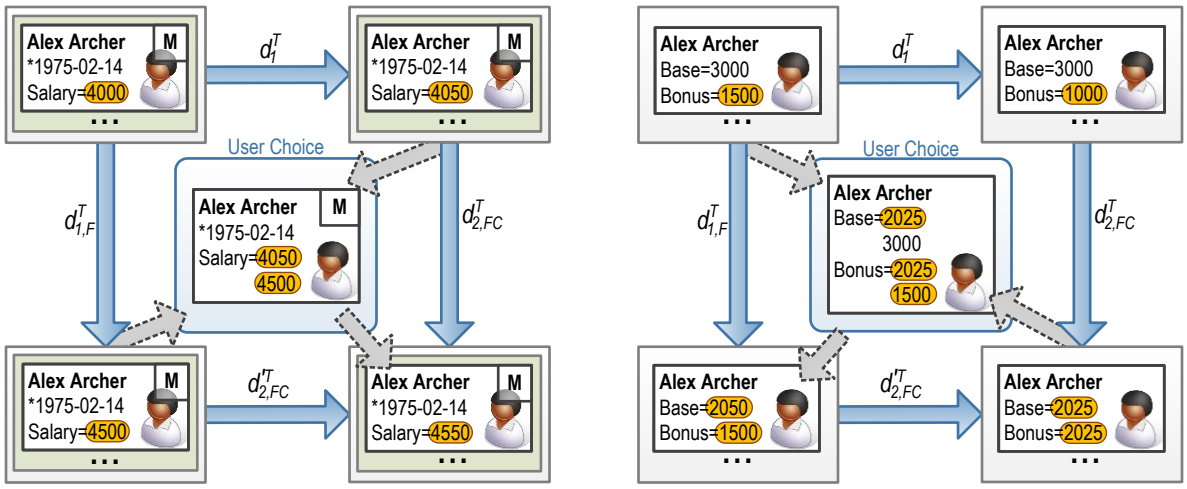


Figure 80: Case 12 - Step 3:Res in fSync operation on the left and in bSync operation on the right

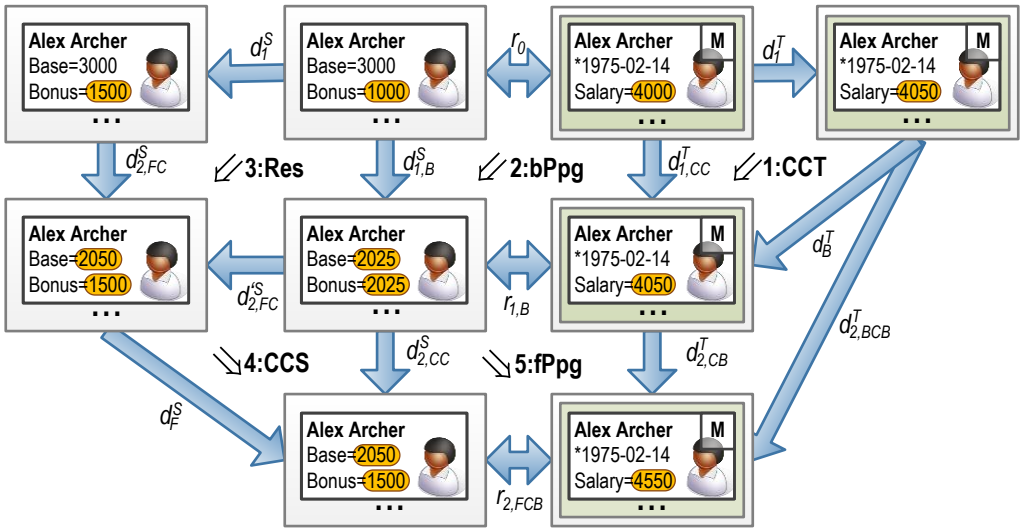


Figure 81: Case 12 - bSync operation

Example 6.13 (Case 13 - Simultaneously resign Jack Judge and increase his salary)**.** Following a rule that is specific to the marketing department, the base salary of Jack Judge gets increased. This update is performed on the source domain and is reflected by d_1^S (see Fig. 82). Concurrently, Jack Judge wants to leave the company. This modification is reflected in the target domain d_1^T (see Fig. 84).

After forward propagating modification d_1^S to the target domain first in Fig. 82, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the change of Jack Judge's salary. However, modifications d_1^T and $d_{1,F}^T$ are in conflict due to the concurrent modification of the attribute Salary and the deletion of Jack Judge's data record. Thus, the user has to solve the conflict manually in selecting the correct solution. Details of the manual conflict resolution step (3:Res) of the fSync operation are illustrated on the left-hand side of Fig. 83. In this

case, the human resource manager selects the increase of the Salary. Afterwards, modification $d^T_{2,CC}$ is backward propagated to the source domain leading to modification $d^S_{2,CB}$ with modification sequence $G^S_0 \overset{d^S_{1,CC}}{\Longrightarrow} G^S_{1,C} \overset{d^S_{2,CB}}{\Longrightarrow} G^S_{2,FCB}$ preserving the change of the base salary.

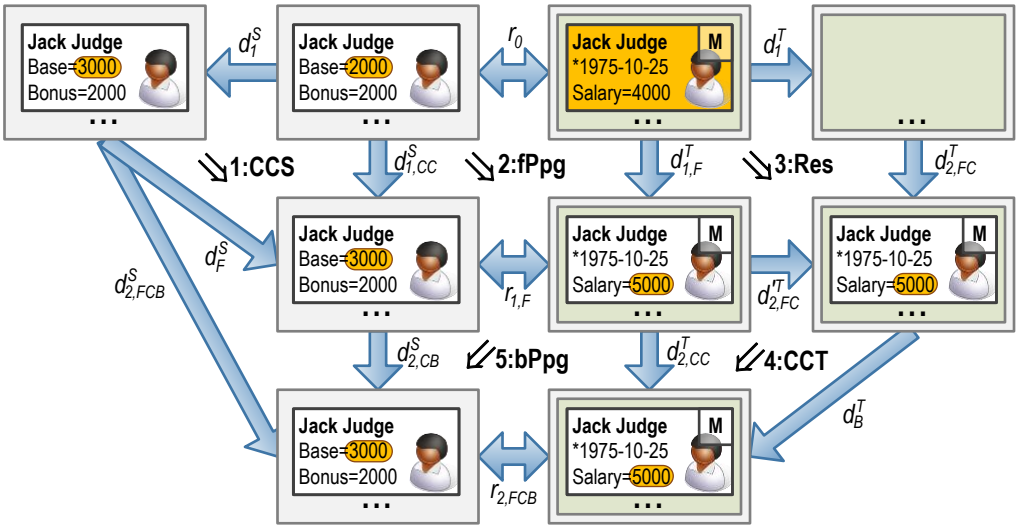


Figure 82: Case 13 - fSync operation

Conversely, after backward propagating modification d^T_1 to the source domain first in Fig. 84, the general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ reflects the deletion of Jack Judges data record. Again, this leads to a conflict of the type attribute modification vs. deletion, so that manual conflict resolution is necessary (similar to the fSync operation). For details of the manual conflict resolution see the right-hand side of Fig. 83. After selecting a solution, modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ reflecting the change of Jack Judges base salary.

Synchronisations fSync and bSync lead to different synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$. In bSync operation, the birth date cannot be restored, by fPpg operation, because of the deletion of Jack Judge's complete data record by the bPpg operation, the information about the correct birth data gets lost in the target domain (see Fig. 84). In contrast, if the prioritise the fPpg operation, we will not lose any information (see Fig. 82).

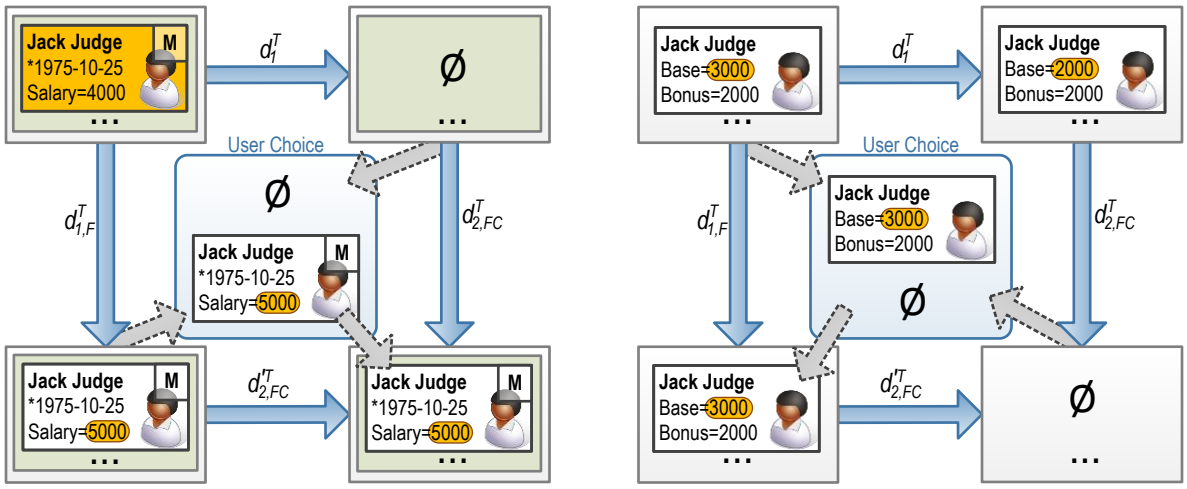


Figure 83: Case 13 - Step 3:Res in fSync operation on the left and in bSync operation on the right

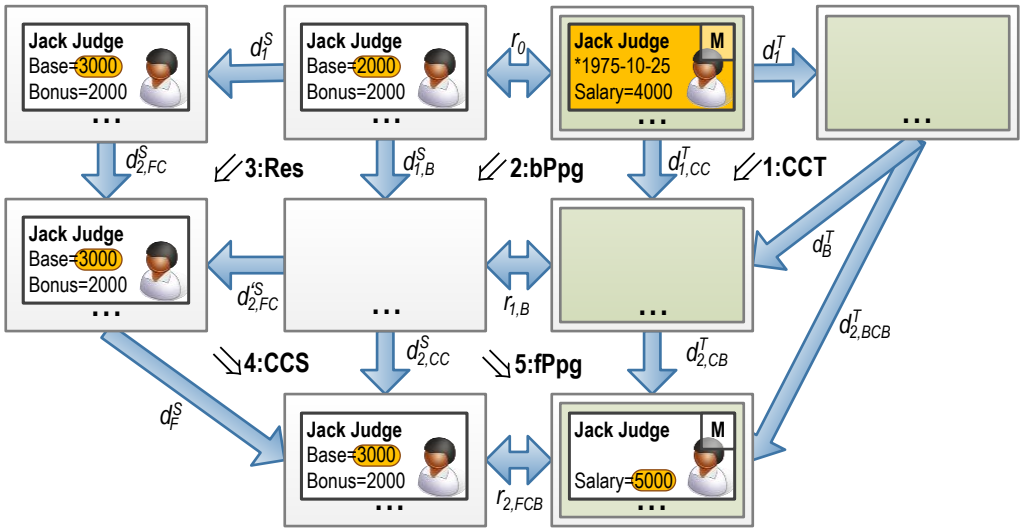


Figure 84: Case 13 - bSync operation

Example 6.14 (Case 14 - Simultaneously discharge Tony Taylor and change his birth date)**.** Tony Taylor plans to leave the company next month. This change is performed in the source model and is reflected by modification d_1^S (see Fig. 87). At the same time, another Tony Taylor is hired by the marketing department, which leads to a misunderstanding and therefore, to an adapt of the birth date. This update is performed in the target model and is reflected by modification d_1^T (see Fig. 85).

After forward propagating modification d_1^S to the target domain first in Fig. 87, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the deletion of Tony Taylor's data record. However, modifications d_1^T and $d_{1,F}^T$ are in conflict due to the concurrent deletion and modification of the attribute Birth. Thus, the user has to solve the conflict manually in selecting either the deletion or the attribute modification. Both possibilities are suggested

by the algorithm. The user decides to keep the data record of Jack Judge and to apply the change of the birth date. Details of the manual conflict resolution step (3:Res) of the fSync operation are illustrated in Fig. 86. After selecting the correct case, modification $d^T_{2,CC}$ is backward propagated to the source domain leading to modification $d^S_{2,CB}$ with modification sequence $G^S_0 \overset{d^S_{1,CC}}{\Longrightarrow} G^S_{1,C} \overset{d^S_{2,CB}}{\Longrightarrow} G^S_{2,FCB}$ representing the change of the birth date.

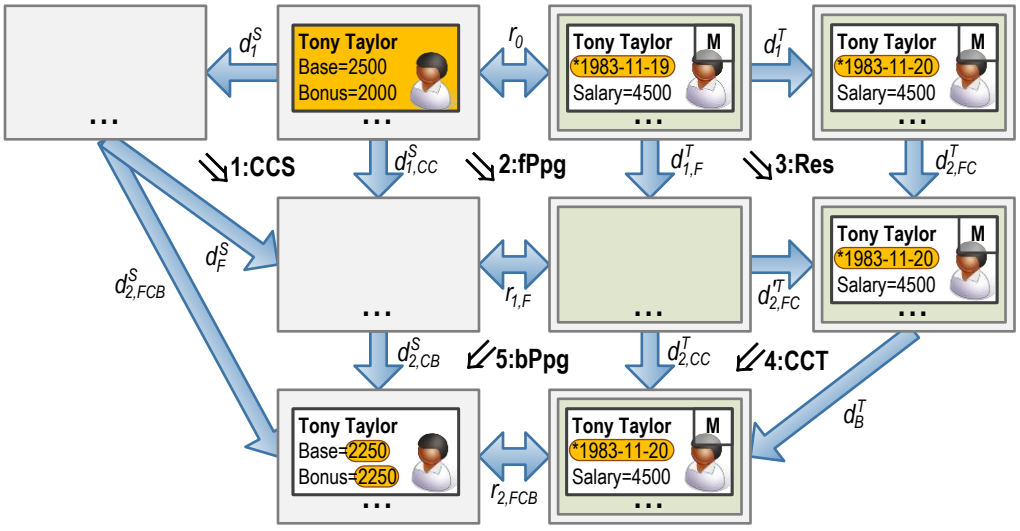


Figure 85: Case 14 - fSync operation

Conversely, after backward propagating modification d^T_1 to the source domain first in Fig. 85, the general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ reflects the deletion of Tony Taylor's data record. Step 3:Res is conflict-free, so modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ deleting Jack Judge's data record.

Synchronisations fSync and bSync lead to different synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$. If we consider fSync in Fig. 87, and the the chosen manual conflict resolution in Fig. 86, then the original detailed information about the correct base and bonus salary is lost, so that the new distribution is calculated by: $Salary = 2 \cdot Base = 2 \cdot Bonus$. In contrast, step 3:Res of bSync is conflict-free and therefore, no manual user input is necessary. Thus, the deletion of Tony Taylor's data record is propagated.

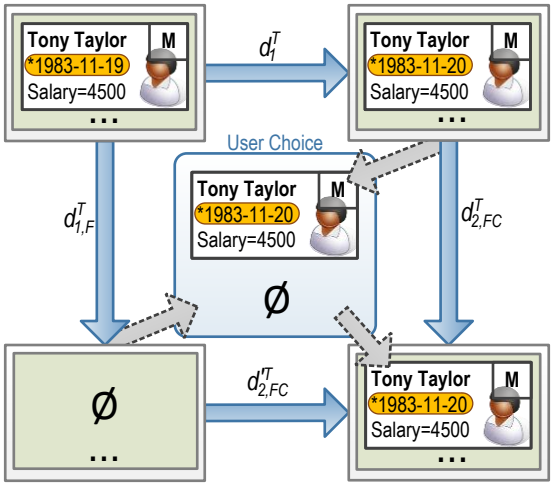


Figure 86: Case 14 - Step 3:Res in fSync operation on the left and in bSync operation on the right

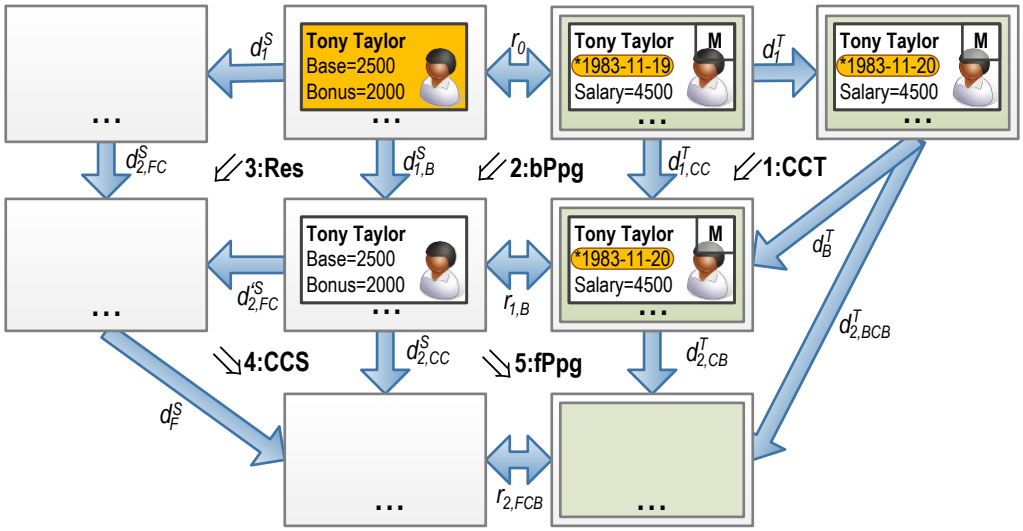


Figure 87: Case 14 - bSync operation

Example 6.15 (Case 15 - Simultaneously hire trainee Lily Lee for a permanent position at the development department)**.** The trainee Lily Lee gets hired for a permanent position at the development department. This change is concurrently entered by modifications d_1^S and d_1^T (see Fig. 90 and Fig. 88). In the source domain, this change is reflected by the deletion of the whole date record. In the target domain, an edge between Lily Lee's node and the development department node is created.

After forward propagating modification d_1^S to the target domain first in Fig. 90, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the deletion of Lily Lee. However, modifications d_1^T and $d_{1,F}^T$ are in deletion-addition conflict. Thus, the user solves the conflict manually in choosing the correct case out of both given possibilities, i.e., the establishment of the membership of Lily Lee to the development department. Details of the manual conflict

resolution step (3:Res) of the fSync operation are illustrated in Fig. 89. After selecting the correct last name, modification $d^T_{2,CC}$ is backward propagated to the source domain leading to modification $d^S_{2,CB}$ with modification sequence $G^S_0 \overset{d^S_{1,CC}}{\Longrightarrow} G^S_{1,C} \overset{d^S_{2,CB}}{\Longrightarrow} G^S_{2,FCB}$ reflecting the addition of the edge between Lily Lee and the development department.

Conversely, after backward propagating modification d^T_1 to the source domain first in Fig. 88, the general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ reflects Lily Lee's hiring at the development department. Step 3:Res is conflict-free, so that modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ preserving the change of Lily Lee's employment status.

Synchronisations fSync and bSync lead to the same synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$, but only if the manual conflict resolution is performed for fSync as illustrated.

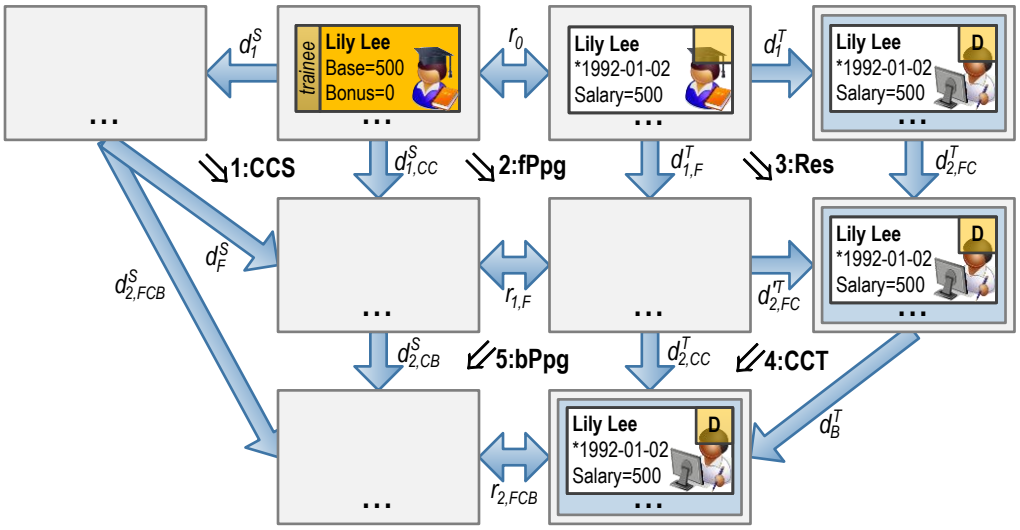


Figure 88: Case 15 - fSync operation

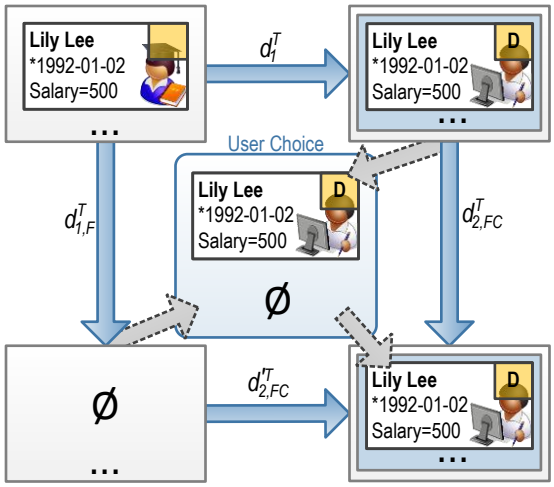


Figure 89: Case 15 - Step 3:Res in fSync operation

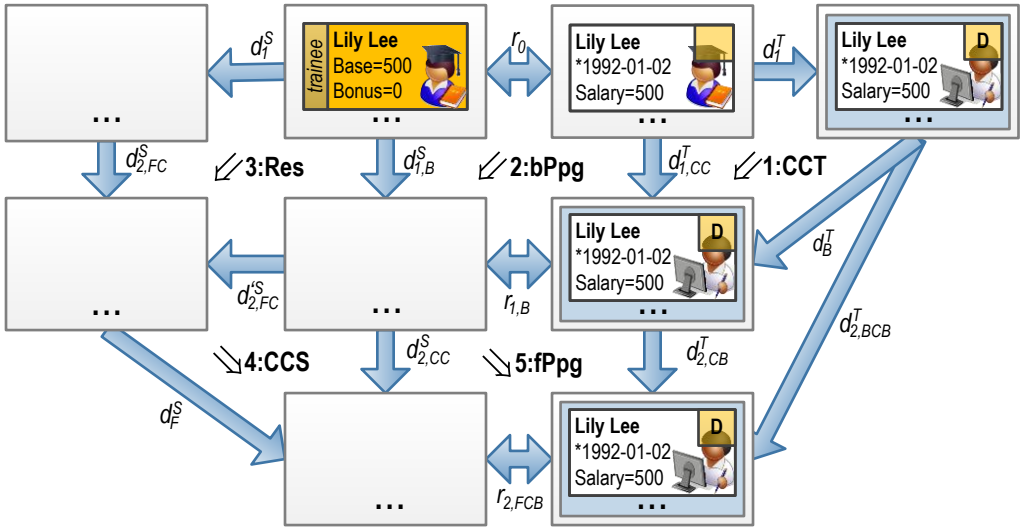


Figure 90: Case 15 - bSync operation

Example 6.16 (Case 16 - Simultaneously hire trainee Willy Wilson for a permanent position at the marketing department)**.** The trainee Willy Wilson gets hired for a permanent position at the marketing department. This change is concurrently entered by modifications d_1^S and d_1^T (see Fig. 92 and Fig. 91).

After forward propagating modification d_1^S to the target domain first in Fig. 92, the general merge construction of the conflicting modifications $d_1^T, d_{1,F}^T$ leads to an already consistent graph $G_{2,FC}^T$ and modification $d'^T_{2,FC}$ with $d'^T_{2,FC} = d_{2,CC}^T$ so that modification sequence $G_0^T \overset{d_{1,F}^T}{\Longrightarrow} G_{1,F}^T \overset{d_{2,CC}^T}{\Longrightarrow} G_{2,FCB}^T$ reflects the hiring of Willy Wilson to the marketing department. Modifications d_1^T and $d_{1,F}^T$ are conflict-free, therefore modification $d_{2,CC}^T$ is backward propagated to the source domain leading to modification $d_{2,CB}^S$ with modification sequence $G_0^S \overset{d_{1,CC}^S}{\Longrightarrow} G_{1,C}^S \overset{d_{2,CB}^S}{\Longrightarrow} G_{2,FCB}^S$ representing the change of Willy Wilson's position.

Conversely, after backward propagating modification d_1^T to the source domain first in Fig. 88, the

general merge construction of the conflicting modifications $d^S_1, d^S_{1,B}$ leads to an already consistent graph $G^S_{2,BC}$ and modification $d'^S_{2,FC}$ with $d'^S_{2,FC} = d^S_{2,CC}$ so that modification sequence $G^S_0 \overset{d^S_{1,B}}{\Longrightarrow} G^S_{1,B} \overset{d^S_{2,CC}}{\Longrightarrow} G^S_{2,BCF}$ reflects the change of Willy Wilson's position. Again, the modifications d^S_1 and $d^S_{1,B}$ are conflict-free, so that modification $d^S_{2,CC}$ is forward propagated to the target domain leading to modification $d^T_{2,CF}$ with modification sequence $G^T_0 \overset{d^T_{1,CC}}{\Longrightarrow} G^T_{1,C} \overset{d^T_{2,CF}}{\Longrightarrow} G^T_{2,BCF}$ representing the hiring of Willy Wilson to the marketing department.

Synchronisations fSync and bSync lead to the same synchronisation results $G^S_{2,FCB} \overset{r_{2,FCB}}{\longleftrightarrow} G^T_{2,FCB}$ and $G^S_{2,BCF} \overset{r_{2,BCF}}{\longleftrightarrow} G^T_{2,BCF}$ with $G^S_{2,FCB} = G^S_{2,BCF}, G^T_{2,FCB} = G^T_{2,BCF}$ and $r_{2,FCB} = r_{2,BCF}$.

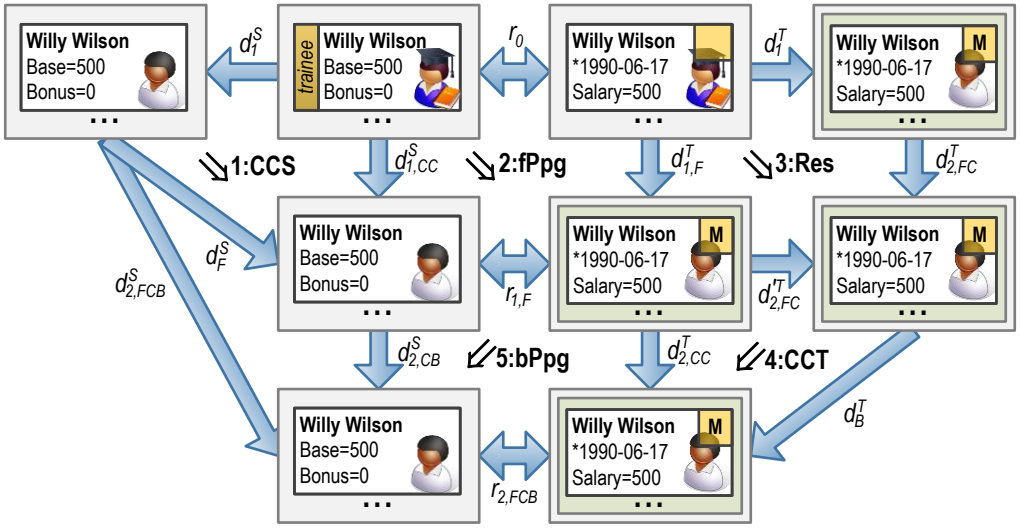


Figure 91: Case 16 - fSync operation

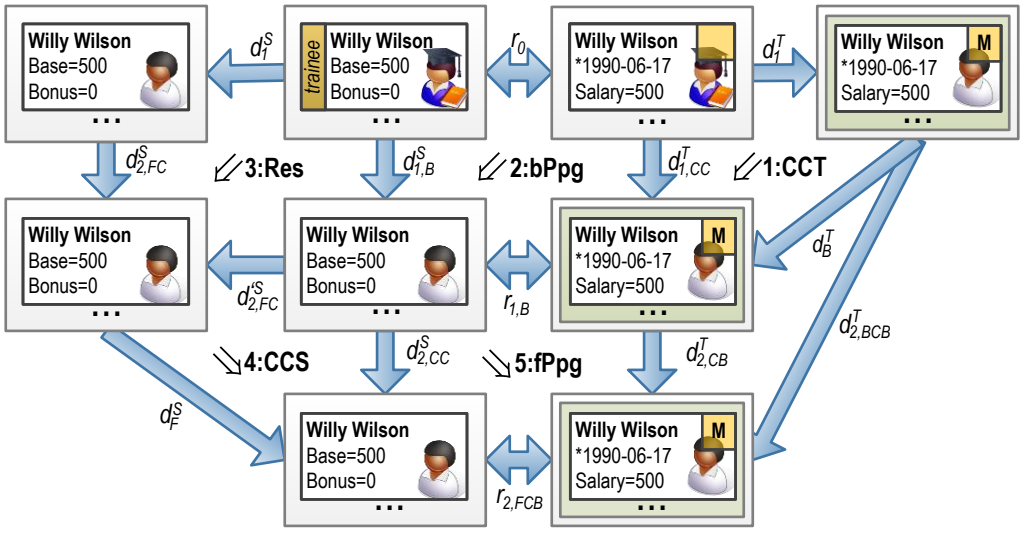


Figure 92: Case 16 - bSync operation

7 Related Work

Triple Graph Grammars were introduced in [25] and since then have been applied successfully, among others, for (concurrent) model synchronisation [7, 15, 12, 18] using the generated operations for bidirectional model transformations [26, 13]. The (concurrent) model synchronisation approach we use in this paper is inspired by the symmetric delta lens (sd-lens) approach introduced in [2].

Several works focus on correctness properties and functional behaviour of the model synchronisation based on triple graph grammars [15, 12, 22, 18]. In [6], a categorical merge construction for two conflicting model updates is given. In [12], a general synchronisation framework for concurrent model updates is given using the results from [6] for resolving conflicts between concurrent model updates. In this work, we extend the concept of filter NACs, which were introduced in [13] for model transformation, to concurrent model transformation.

In [28], a general framework for the synchronisation of concurrent updates is proposed, which requires conflict-free updates as input. The authors introduce requirements, which should hold for all bidirectional model updates, namely consistency, stability and preservation. These requirements are in close correspondence with the laws of correctness and identity ensured for the synchronisation framework in this present paper. Furthermore, an algorithm for model synchronisation is proposed based on model difference approaches. In our scenario, model updates are kept explicit, such that there is no need for model difference computations.

To maintain data consistency, active database systems utilise the event-condition-action (ECA) paradigm to propagate data updates between interrelated data domains by applying ECA rules [24]. An ECA rule defines an event, a condition and an action which is performed once an instance of the event is detected and the condition holds. In active database systems, ECA rules are used to specify and trigger manipulations of data in one domain in response to data updates of another domain. In contrast to the ECA approach where ECA rules define both, expected data updates and the data manipulations for obtaining consistency, our approach enables us to define them separately. TGGs are used to define sets of consistent integrated models. Model updates are defined separately and propagated by applying forward (or backward) propagation operations.

In [21] an efficient control algorithm for bidirectional model transformation based on triple graph grammars is introduced and its correctness, completeness and efficiency is proven. The idea of the control algorithm is to determine dependencies of forward rules (and backward rules, respectively) in order to reduce non-deterministic behaviour in selecting the appropriate rule sequence for the transformation. This work is extended to model synchronisation in [22]. In both works, this algorithm either provides correct models as result or an error. They do not provide strategies for resolving conflicts caused by concurrent updates (operation Res) or models that become inconsistent with respect to the TGG (operations CCS and CCT).

8 Conclusion

In this technical report, we have shown how concurrent modifications in source and target models that are linked by a TGG can be synchronised. More precisely, we have first introduced a non-deterministic concurrent synchronisation framework generalising the existing approach [12] to arbitrary TGGs. Then, we have shown user customisations of the synchronisation process, where the user may choose in general, if the source model should take precedence by using forward synchronisation or, vice versa, the target model through backward synchronisation. Furthermore, we have used filter NACs to improve the efficiency of the forward and backward propagations avoiding backtracking in cases, where parts of the models would remain untranslated. Forward and backward synchronisation may still lead to conflict situations, where the user can select the most adequate resolution.

We already have successfully applied TGGs and Henshin [1, 10] in a large-scale industrial project with the satellite operator SES for the translation of satellite control procedures between different programming languages (see [23], pages 14-15). The satellite Astra 2F is the first satellite running on the translated software and is operational in space since 2012. In future work, we will apply the presented concepts to synchronisation case studies in this field. Particularly, we apply TGGs to visualise and generate source code of satellite procedures at SES. The presented concepts will be utilised to synchronise the source code and its visualisations.

References

[1] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause & Gabriele Taentzer (2010): *Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations*. In Dorina C. Petriu, Nicolas Rouquette & Øystein Haugen, editors: *Model Driven Engineering Languages and Systems - 13th International Conference (MODELS 2010)*, *Lecture Notes in Computer Science* 6394, pp. 121–135. Available at http://dx.doi.org/10.1007/978-3-642-16145-2_9.

[2] Zinovy Diskin, Yingfei Xiong & Krzysztof Czarnecki (2010): *From State- to Delta-Based Bidirectional Model Transformations*. In: *Proc. ICMT'12*, *LNCS* 6142, Springer, pp. 61–76.

[3] H. Ehrig, C. Ermel, F. Hermann & U. Prange (2009): *On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars*. In: *Proc. MODELS'09*, *LNCS* 5795, Springer, pp. 241–255.

[4] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann & Gabriele Taentzer (2007): *Information Preserving Bidirectional Model Transformations*. In: *Proc. FASE'07*, *LNCS* 4422, Springer, pp. 72–86.

[5] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange & Gabriele Taentzer (2006): *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theor. Comp. Science, Springer. Available at http://www.springer.com/3-540-31187-4.

[6] Hartmut Ehrig, Claudia Ermel & Gabriele Taentzer (2011): *A Formal Resolution Strategy for Operation-Based Conflicts in Model Versioning Using Graph Modifications*. In: *Proc. FASE'11*, *LNCS* 6603, Springer, pp. 202–216. Available at http://dx.doi.org/10.1007/978-3-642-19811-3_15.

[7] Holger Giese & Robert Wagner (2009): *From model transformation to incremental bidirectional model synchronization*. *Software and Systems Modeling* 8, pp. 21–43. Available at http://dx.doi.org/10.1007/s10270-008-0089-9. 10.1007/s10270-008-0089-9.

[8] Ulrike Golas, Hartmut Ehrig & Frank Hermann (2011): *Formal Specification of Model Transformations by Triple Graph Grammars with Application Conditions*. *Electronic Communications of the EASST* 39. Available at http://journal.ub.tu-berlin.de/index.php/eceasst/issue/view/23.

[9] J. Greenyer & E. Kindler (2010): *Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars*. *Software and Systems Modeling (SoSyM)* 9(1), pp. 21–46.

[10] (2013): *EMF Henshin – Version 0.9.6*. Available at http://www.eclipse.org/henshin/.

[11] Frank Hermann, Hartmut Ehrig, Claudia Ermel & Fernando Orejas (2011): *Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars - Extended Version*. Technical Report 2011/14, TU Berlin. Available at http://www.eecs.tu-berlin.de/menue/forschung/forschungsberichte/.

[12] Frank Hermann, Hartmut Ehrig, Claudia Ermel & Fernando Orejas (2012): *Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars*. In Juan de Lara & Andrea Zisman, editors: *Int. Conf. on Fundamental Approaches to Software Engineering (FASE'12)*, *Lecture Notes in Computer Science* 7212, Springer, pp. 178–193.

[13] Frank Hermann, Hartmut Ehrig, Ulrike Golas & Fernando Orejas (2010): *Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars*. In J. Bézivin, R.M. Soley & A. Vallecillo, editors: *Proc. Int. Workshop on Model Driven Interoperability (MDI'10)*, MDI '10, ACM, New York, NY, USA, pp. 22–31, doi:http://doi.acm.org/10.1145/1866272.1866277. Available at http://tfs.cs.tu-berlin.de/publikationen/Papers10/HEGO10.pdf.

[14] Frank Hermann, Hartmut Ehrig, Ulrike Golas & Fernando Orejas (2010): *Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars - Extended Version*. Technical Report 2010/13, Technical University of Berlin. Available at http://www.eecs.tu-berlin.de/fileadmin/f4/TechReports/2010/tr_2010-13.pdf.

[15] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin & Yingfei Xiong (2011): *Correctness of Model Synchronization Based on Triple Graph Grammars*. In Jon Whittle, Tony Clark & Thomas Khne, editors: *ACM/IEEE 14th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'11)*, *LNCS* 6981, Springer, pp. 668–682, doi:10.1007/978-3-642-24485-8_49. Available at http://dx.doi.org/10.1007/978-3-642-24485-8_49.

[16] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin & Yingfei Xiong (2011): *Correctness of Model Synchronization Based on Triple Graph Grammars*. In Jon Whittle, Tony Clark & Thomas Kühne, editors: *Model Driven Engineering Languages and Systems*, *Lecture Notes in Computer Science* 6981, Springer Berlin / Heidelberg, pp. 668–682. Available at http://dx.doi.org/10.1007/978-3-642-24485-8_49. 10.1007/978-3-642-24485-8_49.

[17] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin & Yingfei Xiong (2011): *Correctness of Model Synchronization Based on Triple Graph Grammars - Extended Version*. Technical Report TR 2011-07, TU Berlin, Fak. IV.

[18] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, Yingfei Xiong, Susann Gottmann & Thomas Engel (2013): *Model synchronization based on triple graph grammars: correctness, completeness and invertibility*. *Software & Systems Modeling*, pp. 1–29, doi:10.1007/s10270-012-0309-1. Available at http://dx.doi.org/10.1007/s10270-012-0309-1.

[19] Frank Hermann, Hartmut Ehrig, Fernando Orejas & Ulrike Golas (2010): *Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars*. In: *Proc. Intern. Conf. on Graph Transformation (ICGT' 10)*, *LNCS* 6372, Springer, pp. 155–170.

[20] E. Kindler & R. Wagner (2007): *Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios*. Technical Report TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany.

[21] M. Lauder, A. Anjorin, G. Varr & A. Schürr (2012): *Bidirectional Model Transformation with Precedence Triple Graph Grammars*. In: *Proc. ECMFA'12*, *LNCS* 7349, Springer, pp. 287–302.

[22] M. Lauder, A. Anjorin, G. Varr & A. Schürr (2012): *Efficient Model Synchronization with Precedence Triple Graph Grammars*. In: *Proc. ICGT'12*, *LNCS* 7562, Springer, pp. 401–415.

[23] Björn Ottersten & Thomas Engel: *Interdisciplinary Centre for Security, Reliability and Trust - Annual Report 2011*. University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust (SnT). `http://www.uni.lu/content/download/52106/624943/version/1/file/SnT_AR2011_final_web.pdf`.

[24] Adrian Paschke & Alexander Kozlenkov (2009): *Rule-Based Event Processing and Reaction Rules*. In Guido Governatori, John Hall & Adrian Paschke, editors: *Rule Interchange and Applications*, *Lecture Notes in Computer Science* 5858, Springer Berlin Heidelberg, pp. 53–66, doi:10.1007/978-3-642-04985-9_8. Available at `http://dx.doi.org/10.1007/978-3-642-04985-9_8`.

[25] A. Schürr (1994): *Specification of Graph Translators with Triple Graph Grammars*. In: *Int. Workshop on Graph-Theoretic Concepts in Computer Science*, *LNCS* 903, Springer, pp. 151–163.

[26] Andy Schürr & Felix Klar (2008): *15 Years of Triple Graph Grammars*. In: *Proc. Int. Conf. on Graph Transformation (ICGT 2008)*, Springer Verlag, pp. 411–425, doi:10.1007/978-3-540-87405-8_28. Available at `http://dx.doi.org/10.1007/978-3-540-87405-8_28`.

[27] TFS-Group, TU Berlin (2013): *AGG*. `http://www.tfs.tu-berlin.de/menue/forschung/software/parameter/en/`.

[28] Y. Xiong, H. Song, Z. Hu & M. Takeichi (2011): *Synchronizing concurrent model updates based on bidirectional transformation*. *Software and Systems Modeling*, pp. 1–16.

[29] Yingfei Xiong, Hui Song, Zhenjiang Hu & Masato Takeichi (2009): *Supporting Parallel Updates with Bidirectional Model Transformations*. In: *Proc. ICMT'09*, Springer, pp. 213–228.