On Dealing With Semantically Conflicting Business Process Changes*

Stefanie Rinderle Manfred Reichert
Peter Dadam
University of Ulm, Faculty of Computer Science,
Dept. Databases and Information Systems
{rinderle, reichert, dadam}@informatik.uni-ulm.de

Abstract

Correct propagation of process type changes to long-running process instances and the capability to perform ad-hoc-modifications of individual process instances are essential requirements for any process management software. In particular, in many cases it becomes necessary to propagate process type changes to individually modified process instances as well. In doing so, one must not only enable state-related compliance checks, but is additionally confronted with structural and semantical conflicts that may exist between process type and process instance changes. For the first time, this paper identifies and classifies semantical conflicts between process type and instance changes, and illustrates them by sophisticated examples. In order to be able to adequately deal with semantical conflicts at change propagation time we provide formal methods for conflict detection and discuss strategies to deal with different kinds of semantically conflicting changes.

1 Introduction

The use of enterprise-wide standard software (e.g., ERP systems) often leads to a rigid implementation of the business processes. Respective software systems lack the necessary flexibility since process adaptations are expensive, time-consuming and error-prone. However, the ability to quickly react to business process changes and to adapt process-oriented application systems accordingly is indispensable for any enterprise to stay competitive in its market [30, 9, 24]. Such process changes range from ad-hoc modifications of single process instances to evolutionary changes of the process schema itself [30]. Ad-hoc changes (e.g., performance of an unplanned activity for a particular business case) become necessary, for instance, to deal with real-world exceptions, and they lead to individually modified process instances [21]. A process type or –

*This work was done within the research project "Change management in adaptive workflow systems", which has been founded by the German Research Community (DFG).

more precisely – its related schema has to be modified when new laws come into effect, optimized or restructured business processes are to be implemented, or rapid reactions to a changed market are required. In this context, related process instances may have to be adapted as well [30, 6].

The challenging question is how to efficiently and correctly *propagate* process schema changes to a potentially large number of (long-)running process instances. To smoothly *migrate* process instances to a changed schema is already a non-trivial problem if *unbiased* process instances, i.e., instances without individual modification, are concerned [30, 25, 1, 11, 23]. In this case, we are "only" confronted with a multitude of concurrently running process instances being in different execution states or in different loop iterations but all refering to the same process schema. The "migration-problem" becomes more complicated by orders of magnitude if the concerned process instances have been individually modified (previous to the process type change).

One problem arising in this context is that process schema changes may be in conflict with the previous process instance changes. For existing approaches dealing with adaptive processes this problem has not been addressed so far since an ad-hoc change of a single process instance always leads to a new process schema version [19, 33]. Thus, once a process instance has been individually modified, it may be no longer adaptable to future business process changes or – more precisely – to future process schema changes at the type level. However, doing so is far away from practical requirements, in particular when dealing with long-running process instances. As an example take a medical treatment process in a hospital. Even if a process instance related to a particular patient is individually modified (e.g., by performing an unplanned lab test) it should also take benefit from future business process optimizations (i.e., changes at the process type level). Otherwise, the process oriented information system will not be accepted by users.

1.1 Problem Description

Concerning the propagation of process schema changes to individually modified instances, in general, we have to cope with **state related**, **structural**, and **semantical conflicts** [23, 22]. When propagating changes at the process type level to instances still running according to their original process schema (unbiased instances) the most important thing we have to care about is that the current instance state does not conflict with the respective change at the schema level (e.g., an already completed activity must not be deleted at the instance level). Change propagation therefore leads to two sets of instances – those which are compliant with the changed schema and those which are not [23, 22]. When propagating schema changes to individually modified instances, however, things become more complicated. First of all, the treatment of structural conflicts between schema and instance changes has not been exhaustively addressed in the workflow literature so far (however some work on related problems in the field of cooperative information systems exists, e.g. [31]). Such conflicts may generate, for example, a deadlock-causing cycle due to the uncontrolled insertion of edges or activities on both, the schema and the instance level [23, 22].

Semantic issues have been analyzed, for example, in the area of secure workflows [2]. But so far, there has been a total lack of approaches dealing with issues related to **semantical conflicts between process schema and instance changes**. The adequate treatment of such conflicts is a must for any adaptive process management system. In particular, undesired system behavior or runtime errors caused by the introduction of semantically conflicting changes have to be avoided in any case. A first challenging issue is to identify scenarios in which semantical conflicts between concurrent changes occur. To give an idea we exemplarily describe two of these scenarios in the following. Note that these scenarios are extreme examples ranging from semantical conflicts between structurally overlapping changes (cf. *Example 1*) to changes conflicting at a high semantical level (cf. *Example 2*).

Example 1: Very important and practically relevant are scenarios where instances (or more precisely the actors working on them) **partially** or **totally** anticipate future process schema changes. As an example take Fig. 1. Assume that again and again instances, exemplarily instances I_1 and I_2 in Fig. 1, have diverged from their original process schema since the invoice cannot be correctly generated before packing the goods. Therefore, I_1 and I_2 have been individually modified by changing the order of activities `make invoice` and `pack goods` to "`pack goods` before `make invoice`". After it has been recognized that these instance changes indicate a "weak point" in the related (original) process schema, e.g., by process mining techniques [14], the schema itself is changed by re-ordering the respective activities to "`pack goods` before `make invoice`" (cf. Fig. 1). Though the schema resulting from the change at process schema level and the current instance schemes are the same, the instances cannot be migrated to the new schema version[1]. In more detail, the migration of the individually modified process instances (e.g., I_1 and I_2 in Fig. 1) will fail. Reason is that edges which are to be deleted when adapting the structure of running instances to the changed process schema are no longer present. As an example take the edge '`getOrder` → `makeInvoice`' which is missing for instances I_1 and I_2. Consequently, these instances are excluded from migrating to S'. But just this is a completely undesired system behavior from a semantical point of view.

Another example for a semantical trap as a result of process schema and instance changes is depicted in Fig. 4. Here, in contrast to the example given in Fig. 1, instances are not excluded from migrating to the changed schema. However, migration results in an execution schema where the same activities will be worked on twice although this is not required and also not desired in the given context.

Example 2: A semantical conflict of the other extreme is caused by process schema and instance changes which are **semantically incompatible**. As an example take Fig. 2. Due to an anaphylactic shock of the respective patient instance I_1 was individually modified by inserting activity `allergyDrugY` between activities `examination` and `operation`. After the introduction of a new drug `newDrugX` the underlying process schema `S` is changed by inserting activity `newDrugX` between `examination` and `operation`. Assume that this activity is medically

[1] In this context, migration means to re-link the instances from their current to the new schema version. In the given situation this could be achieved without propagating changes.

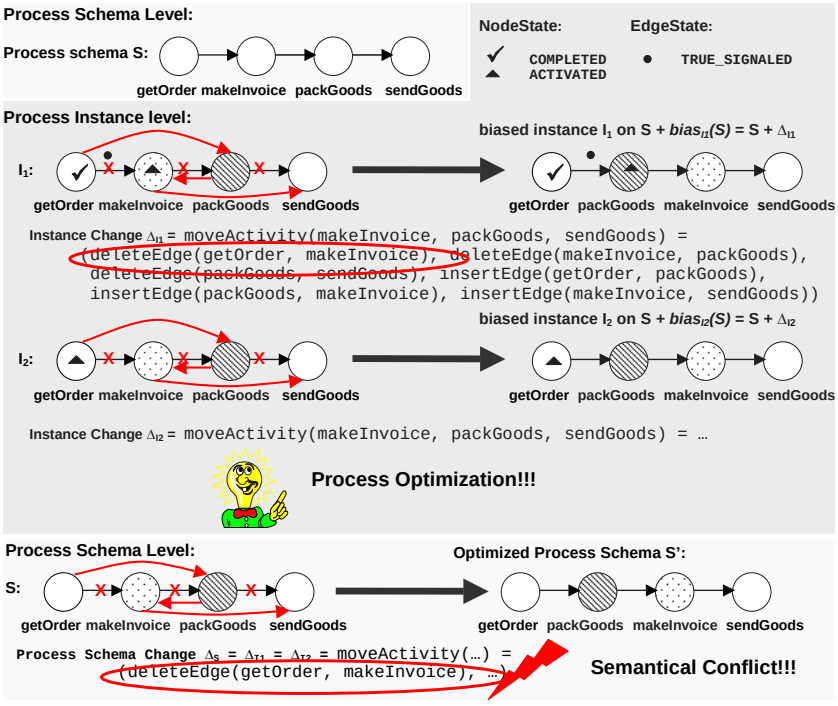


Figure 1: *Semantical Conflict: Instances Anticipating a Process Schema Optimization*

not compatible with `allergyDrugY` due to an undesired drug interaction. Propagating this process schema change to the already modified instance I_1 would cause no problems regarding structural and state related conflicts as it can be easily seen from Fig. 2. But from a semantical point of view, both process schema and instance changes are non-compliant due to the mentioned medical incompatibility between the two drugs. Obviously, to solve this important problem we need a lot of semantical knowledge about the changes to be applied.

There are further interesting examples (cf. Section 3). However, the presented ones already show that there is a high need for an intelligent treatment of semantically conflicting changes.

1.2 Contribution

In this paper, we make the following contributions.

1. For the first time, we present a comprehensive *classification of semantical conflicts* between process schema and instance changes potentially leading to an unintended (semantical) system behavior if the changes are supplied in an uncontrolled manner. To illustrate the

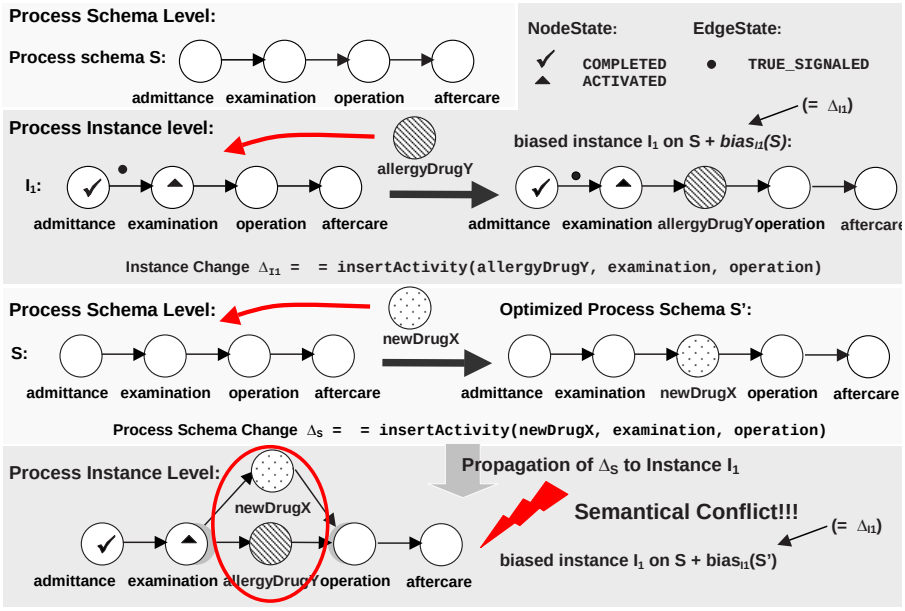


Figure 2: *Semantically Non-Compliant Schema and Instance Changes*

different kinds of semantical conflicts we provide sophisticated examples of high practical relevance and discuss specific problems emerging in this context.

2. We present (formal) methods for detecting semantical conflicts that arise when process schema and instance changes partially or totally overlap. In this context, we present an excursion to the field of **process schema and instance isomorphisms**.

3. We show how individually modified instances can be migrated to a changed schema version if the corresponding modifications semantically overlap. This is crucial for the implementation of any adaptive process management system.

4. Starting from our classification, we discuss several suitable methods to deal with semantically conflicting process schema and instance changes.

The remainder of the paper is organized as follows: Section 2 discusses related work and summarizes important background information which is helpful for the further understanding of the paper. In Section 3 we present a classification of semantical conflicts and we illustrate them by practical examples. In Section 4 formal methods for detecting semantical conflicts are presented. Section 5 adds methods to concretely deal with these conflicting changes. We close with a summary and an outlook on further issues in Section 6.

2 Related Work and Background Information

There is a multitude of approaches dealing with adaptive processes. The scope ranges from ad-hoc changes of single process instances to evolutionary changes of the process schema [30]. The latter implies the question how to efficiently propagate the process schema changes to a collection of running process instances but without causing inconsistencies or errors in the sequel [30, 1, 6, 11, 19, 23, 25, 32]. Thus, the provision of suitable *correctness criteria* is indispensable [23].

One of the first approaches providing a generic correctness criterion for process schema change propagation (the so called *compliance criterion*) has been given in the WIDE project [6]. This criterion is suitable for process models which log the previous execution of process instances in *execution histories*. Then a process instance I (created from schema S) is *compliant* with a changed process schema S' if the execution history of I can be correctly replayed on S'. This criterion works well as long as no cyclic process structures are taken into account [23]. In conjunction with loops, however, the compliance criterion is too restrictive since it may needlessly exclude process instances from migrating to a changed process schema. Furthermore efficiency issues with respect to compliance checking have not been addressed.

The compliance criterion is used by several other approaches as well: BREEZE [26, 27, 25] provides nice strategies how to deal with process instances which are not compliant with the changed process schema. Furthermore, BREEZE does not only focus on control flow changes, but deals with a broad spectrum of process modifications when comparing it to other approaches (incl. temporal aspects [25]). TRAMs [19] provides concepts for managing different versions of a process schema and their running process instances. Furthermore, it cares about how to check compliance of running instances with a changed process schema. For this purpose, each change operation is equipped with a *migration condition*, which enables the runtime system to argue about compliance of the respective process instances.

As opposed to these history-based approaches there are solutions where only the actual "state tokens" of a process instance can be determined (e.g., Petri-Net based approaches). Consequently, these approaches cannot directly use the compliance criterion as described above. In [11, 10], consistency can be only ensured for special change operations. This implies the construction of a "hybrid" process schema which reflects parts of both, the old and the changed process schema. The definition of the rules to map tokens from the old to the new net has to be manually carried out by the user. Actual results from the Petri-Net community come from [30, 29]. The authors propose *branching bisimularity* as a correctness criterion for process schema evolution. Informally, a process instance I can migrate to a changed process schema S' if each action of I can be simulated on S' as well. Unfortunately, branching bisimularity can only be ensured for special change operations, e.g., the insertion or deletion of parallel or alternative branches. Order-changing operations like swapping or parallelizing of activities are dropped out by this criterion. Apart from this, Petri-Net based approaches [11, 30, 1] lack a clear discussion of issues related to data flow adaptations, adaptations in connection with loops, and the concrete realization of migration procedures (to avoid the so-called "dynamic change bug").

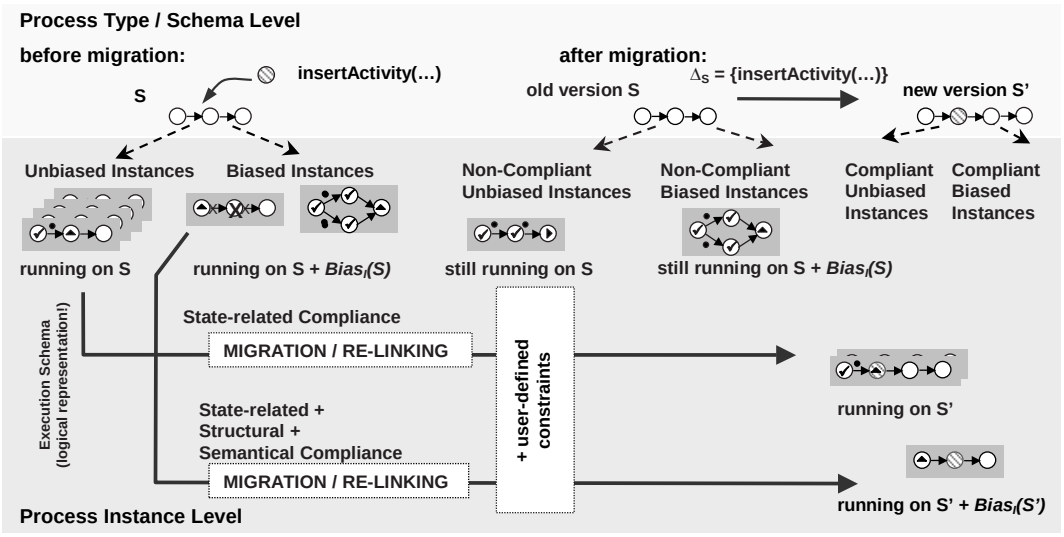


Figure 3: *Migration Process*

There are further approaches, e.g., rule-based [12], object-oriented [15, 32] and statechart-based [13] models. ULTRAflow [12] focuses on modifying the implementation and the semantical information (e.g., compatibility matrices) of workflows. However, ULTRAflow totally factors out several change operations and data flow issues. $WASA_2$ [32] offers a correctness criterion based on the mapping of process instances against process schemes. MOKASSIN [15] realizes process instance changes by encapsulating respective change primitives but does not consider correctness issues at all.

At this point of the discussion we can draft a clear distinction line: All mentioned change propagation approaches can be only used if the concerned process instances have not been individually modified, i.e., if their execution is still based on the original schema from which they were derived. As motivated, however, it is also very important to support the propagation of process schema changes to previously modified process instances as well [18]. While the mentioned correctness criteria only consider the state of process instances when propagating schema changes to them we are now additionally confronted with the problem of dealing with *structural* and *semantical* conflicts between process schema and instance changes (cf. Fig. 3).

Structural conflicts between two (concurrent) changes result when their combined application leads to an undesired or incorrect execution behavior. If, for example, two (concurrent) changes are applied in an uncontrolled manner this might lead to deadlock-causing cycles. Therefore we need a criterion ensuring structural and state related correctness for the combined use of process schema and instance changes. Furthermore, it is fundamental to provide methods for re-linking individually modified (*biased*) instances to a changed process schema. Suitable methods are based on the notion of *commutativity of changes*, i.e., independent from the application order of

changes the resulting instance schema is always the same. A more comprehensive treatment of these questions can be found in [22]. What migration / re-linking concretely means is depicted in Fig. 3: Instances previously assigned to process schema S are now running according to the changed schema S' on condition that they are compliant with S'. Note that – from a logical point of view – each instance has an own execution schema as depicted in Fig. 3. This execution schema results from the original schema S plus the individual bias of I (formally: $bias_I(S)$). Generally, $bias_I(S)$ results from the consecutive execution of all ad-hoc changes applied to I so far (formally: $bias_I(S) := \Delta_1^I \circ \ldots \circ \Delta_k^I$ where $\Delta_1^I, \ldots, \Delta_k^I$ denote individual changes on I). In particular, $bias_I(S)$ is the "difference" between S and the logical execution schema of I. How this execution schema is internally represented and whether it is materialized or not is outside the scope of this paper.

To our best knowledge so far there has been a complete lack of approaches dealing with *semantical conflicts* between process schema and instance changes. This paper makes a first important contribution to close this "gap" and wants to encourage the research community to spend more effort on semantic issues in the context of adaptive process management.

3 Classification Of Schema And Instance Changes

In the following we restrict our considerations to the propagation of schema changes to biased instances (cf. Fig. 3). Therefore, let S be a process schema and let Δ_S be a change which transforms S into another process schema S' at time t_S. Let further $I_1, \ldots, I_n$ be a collection of biased process instances derived from S with biases $bias_{I_1}(S), \ldots, bias_{I_n}(S)$ at time t_S. In order to decide whether $I_1, \ldots, I_n$ can migrate to S' or not, we require several checks regarding state-related, structural, and semantical conflicts. In this paper, we focus on the latter one. Table 1 presents a *general classification* for possible semantical conflicts between schema change Δ_S and instance changes $bias_{I_1}(S), \ldots, bias_{I_n}(S)$. At the presence of such conflicts the uncontrolled propagation of Δ_S to $I_1, \ldots, I_n$ may lead to an undesired (semantical) system behavior or even to runtime errors.

As exemplarily shown in Section 1 semantical conflicts between process schema and instance changes may arise if the changes partially or totally overlap. For totally overlapping changes we use the term *equivalent changes* which is defined in Section 4. For partially overlapping changes, basically, we distinguish between two scenarios: Either the process schema change subsumes the whole process instance change (or vice versa) – then we denote these changes as *subsumption equivalent* – or their intersect is not empty (but both contain additional changes). We call the latter *partially equivalent changes*. Finally, process schema and instance changes can be semantically *different*. All possible kinds of relationships between process schema and instance changes are summarized in Table 1.

In the following, we explain the classification given in Table 1 by providing practical examples for each category. For the sake of comprehensibility we shrunk the collection of relevant instances

Table 1: *Classification Of Semantically Conflicting Changes*

Schema Change Δ_S *and Instance Bias* $bias_{I_k}(S)$ $(k \in \{1, \ldots, n\})$ *may be*

(1) equivalent	*(2) subsumption equivalent*	*(3) partially equivalent*	*(4) different*
$\Delta_S \equiv bias_{I_k}(S)$	(2.1) $\Delta_S \succ bias_{I_k}(S)$	$\Delta_S \cap bias_{I_k}(S) \neq \emptyset$	$\Delta_S \cap bias_{I_k}(S) = \emptyset$
(cf. Fig. 1, 4)	(cf. Fig. 5)	(cf. Fig. 7,8)	(4.1) semantically compliant
	(2.2) $\Delta_S \prec bias_{I_k}(S)$		(4.2) semantically
	(cf. Fig. 6)		non-compliant (cf. Fig. 2)

$I_1, \ldots, I_n$ (with biases $bias_{I_1}(S), \ldots, bias_{I_n}(S)$) to an arbitrary instance I_k $(k = 1, \ldots, n)$. But we always have to be aware that we are possibly confronted with a large number of biased instances which may have to be migrated to a modified schema!

(1) Δ_S **and** $bias_{I_k}(S)$ **are equivalent:** A first example of equivalent changes has been already presented in Section 1 (cf. Fig. 1). Another example is depicted in Fig. 4. Process schema S is modified by schema change Δ_S (the insertion of an additional activity `controlShipment`) after instance change Δ_{I_k} has indicated this process optimization at the instance level. Thus, we have two equivalent changes Δ_S and $bias_{I_k}(S)$ $(= \Delta_{I_k})$. Propagating Δ_S to instance I_k would cause no problem with respect to state as well as structural properties. However, doing so results in a totally unclear semantics: Does the modeler really wants the user to work on activity `controlShipment` twice?

(2) Δ_S **and** $bias_{I_k}(S)$ **are subsumption equivalent:** In addition to equivalent changes we are confronted with *subsumption equivalent* process schema and instance changes. Here, we have to distinguish two cases as depicted in Table 1:

In case (2.1) (cf. Table 1), Δ_S subsumes $bias_{I_k}(S)$ but may include further changes $(\Delta_S \succ bias_{I_k}(S))$. An example is depicted in Fig. 5. Process schema S is modified by schema change Δ_S which re-arranges activities `B` and `C` to the converse order "`B` after `C`". However, at the instance level, Δ_{I_k} has already changed the activity order "`B` after `C`" to "`B` parallel executable to `C`" for instance I_k (and therefore $bias_{I_k}(S) = \Delta_{I_k}$). Having a closer look at $bias_{I_k}(S)$ and Δ_S, we can see that Δ_S subsumes all basic change operations (i.e., all edge insertions / deletions) of $bias_{I_k}(S)$ but is enlarged with additional modifications. Thus we get $bias_{I_k}(S) \prec \Delta_S$. As it can be easily seen, propagating Δ_S on I_k would not be possible at the moment (e.g., edge `A` $\rightarrow$ `B` is no longer present for I_k), but may be necessarily desired.

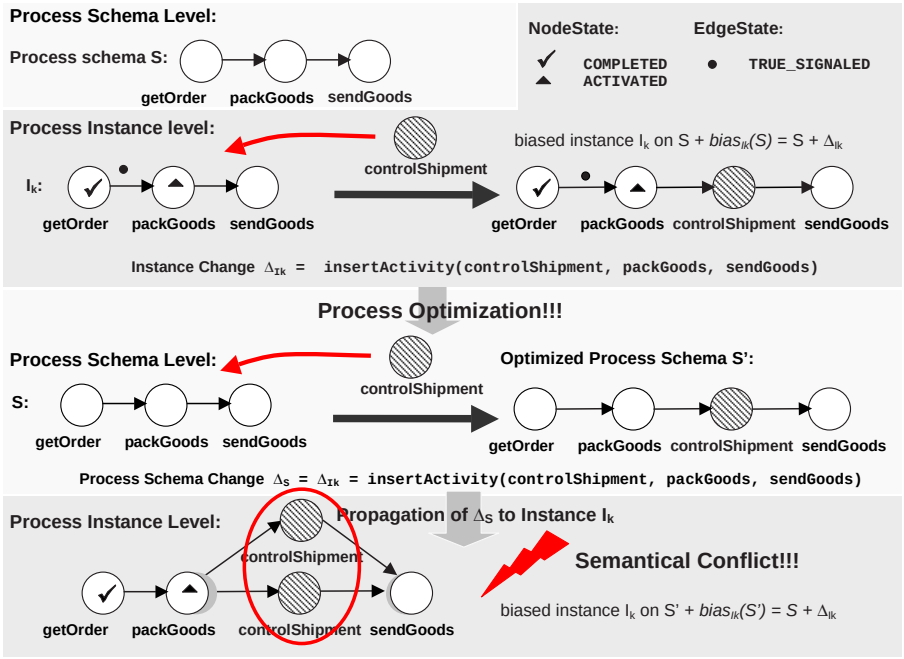


Figure 4: *Semantical Conflict: Double Execution Of Activities*

Case (2.2) in Table 1 occurs if $bias_{I_k}(S)$ has totally anticipated process schema change Δ_S but comprises additional changes ($bias_{I_k}(S) \succ \Delta_S$). This situation arises, for example, if previous modifications at the instance level indicate a schema optimization Δ_S. However, some of the respective instances have been additionally adapted to special customer conditions or exceptional situations as it is depicted in Fig. 6: Here, $\Delta_{I_{k_1}}$ inserted an additional activity `controlGoods` leading to a later process schema optimization Δ_S which also inserts activity `controlGoods`. However, instance I_k has been additionally adapted by change $\Delta_{I_{k_2}}$ to the requirements of a particular customer (therefore $bias_{I_k}(S) = \Delta_{I_{k_1}} \circ \Delta_{I_{k_2}}$). Consequently, $bias_{I_k}$ includes Δ_S but is enlarged. Here, a propagation of Δ_S on I_k is possible but would lead to an undesired double execution of activity `controlGoods`.

(3) Δ_S and $bias_{I_k}(S)$ are partially equivalent ($\Delta_S \cap bias_{I_k}(S) \neq \emptyset$):

As an example take Fig. 7. Here, instance change Δ_{I_k} is carried out in an "ad-hoc-manner", i.e., the newly inserted data element `patWeight` is written by the newly inserted *parameter provision service* [21] (therefore $bias_{I_k}(S) = \Delta_{I_k}$). This service prompts the user for the missing input data and is invoked when activity `medication` is started. Thus, the input parameters of the inserted activities are correctly supplied at runtime. Assume that lifting Δ_{I_k} to the process schema level, the process designer wants data element `patWeight` to be written by a new activity `getWeight` such that the input parameters of activity `medication` are correctly supplied.

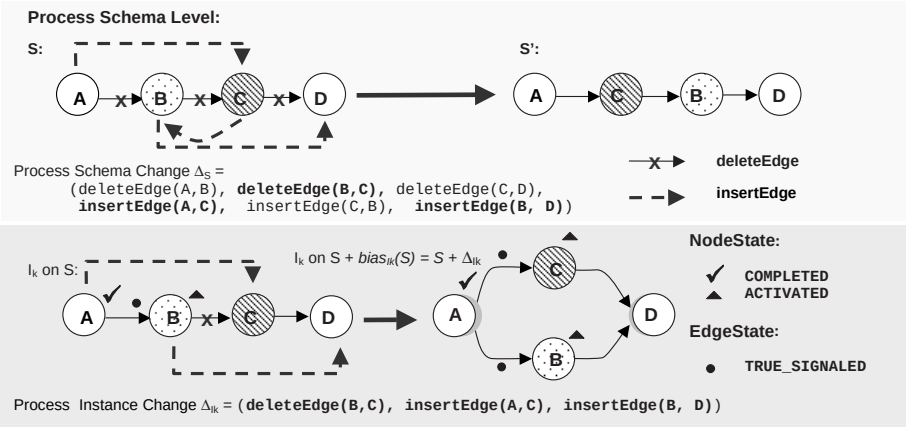


Figure 5: *Process Schema Change Comprises And Enlargens Process Instance Change*

Propagating Δ_S to I_k would therefore result in writing the same data element `patWeight` twice. Here, an interesting variant would be to undo the "provisional" instance change Δ_{I_k} and to replace it by process schema change Δ_S.

Fig. 8 shows another example for partially equivalent changes at the schema and instance level. Instance I_k has been already changed by inserting a new activity `getWeight` between `prepare` and `operation`. Assume that at a later point in time activity `getWeight` is inserted at the proces schema level but at another position in the process graph (namely the upper branch of the parallel branching). Intuitively, propagating Δ_S to I_k is not reasonable since activity `getWeight` would then be executed twice in the sequel. Interestingly, propagating $\Delta_S \neg \Delta_{I_k}$ leads to a hybrid structure of the execution schema of I which reflects both, the process schema change ("`getWeight after prepare`") and the process instance change ("`getWeight after advise`").

(4) Δ_S and $bias_{I_k}(S)$ are different ($\Delta_S \cap bias_{I_k}(S) = \emptyset$):

Case (4.1) where schema change Δ_S and instance bias $bias_{I_k}(S)$ are different and semantically compliant would be the most common one. Here, the propagation of the "whole" change Δ_S is desired if structural and state-related conflicts between Δ_S and $bias_{I_k}(S)$ can be ruled out [22].

In Section 1, we have already presented an interesting example for Case (4.2) where schema change Δ_S and instance bias $bias_{I_k}(S)$ are different and semantically non-compliant. To be able to detect such semantical incompatibilities between process schema and instance changes we first have to formalize the semantics of the respective change. For a basic explanation of our ideas in this context we refer to Section 5.

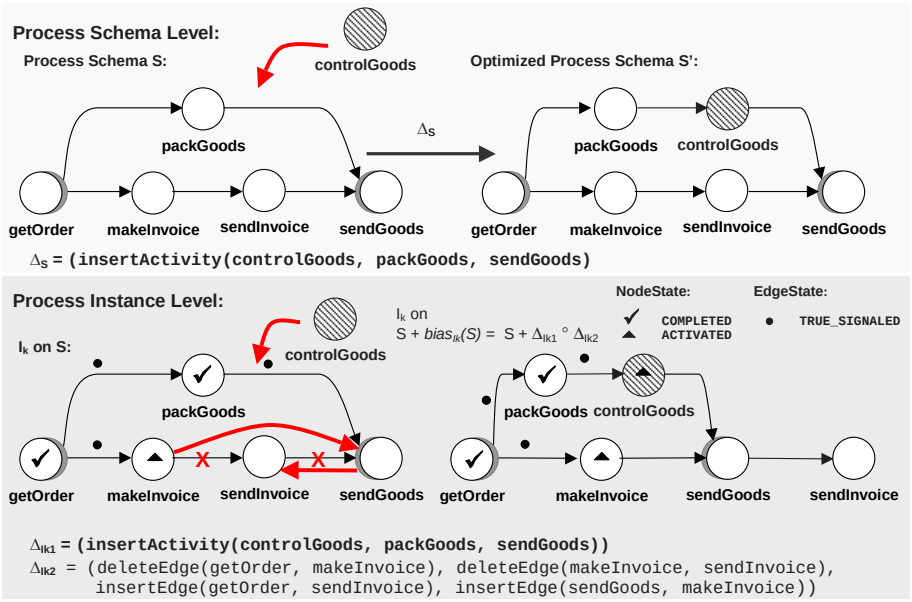


Figure 6: *Process Instance Change Enlargens Process Schema Change*

4 Detecting Semantically Conflicting Change Relations

As described in Table 1 (cf. Section 3) there are different possible relations between process schema and instance changes. Some of these relations may lead to semantically undesired behavior as we have shown by means of examples. Therefore, we need formal criteria to be able to decide whether a process schema change is semantically conflicting with a process instance change or not. More precisely, we need formal criteria to detect the kind of relationship between a process schema and a process instance change to be able to react in a suitable way.

Therefore let S be a process schema and let Δ_1 and Δ_2 be two changes on S: Δ_1 transforms S into schema $S^{(1)}$ and Δ_2 transforms S into schema $S^{(2)}$. As mentioned above we want to decide whether Δ_1 and Δ_2 are equivalent, subsumption equivalent, partially equivalent, or different (cf. Table 1). In principle, there are two possibilities: One is to compare Δ_1 and Δ_2 directly. The other possibility is to compare the resulting process schemes $S^{(1)}$ and $S^{(2)}$ whereas this variant can only be used to detect whether changes are equivalent (cf. Table 1). Thus we start with the direct comparison of the resulting schemes $S^{(1)}$ and $S^{(2)}$ in Section 4.1. Doing so yields to a clear theoretical basis for further considerations. In Section 4.2 we provide the general method of comparing changes Δ_1 and Δ_2 (applicable if Δ_1 and Δ_2 are equivalent, subsumption equivalent, partially equivalent, or different).

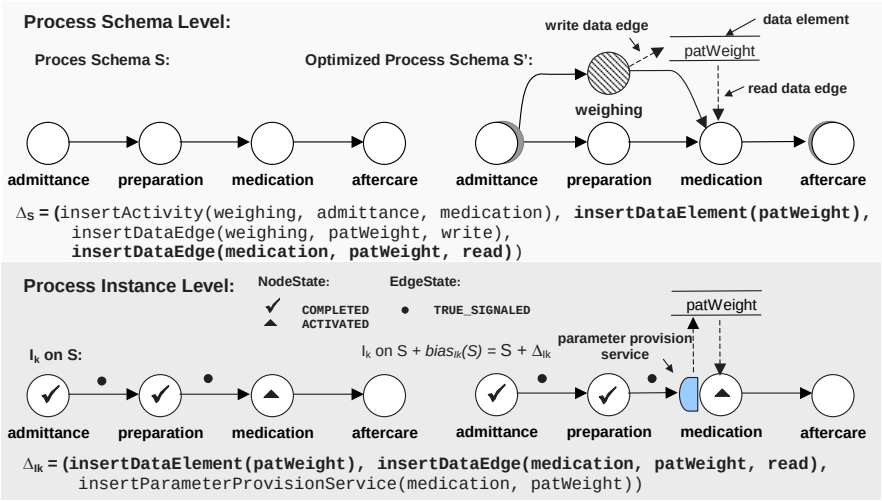


Figure 7: *Partially Equivalent Changes: Insertion of Parameter Provision Services*

4.1 Process Schema Equivalence And Isomorphism

In order to be able to compare two process schemes we first provide a definition for *execution equivalence*. With this we mean that one schema can simulate every executional behavior of the other schema and vice versa. This can be roughly compared to the notion of branching bisimularity for Petri-Net-based approaches as described in [30]. In this context, we make use of the *execution history* H_S which is usually maintained for each process instance of schema S. Similarily, in [2] a notion of semantic equivalence between two workflows is defined based on the semantic projection of their execution histories. Generally, H_S logs start and end events of each process activity and the respective read and write accesses on process data elements.

Definition 1 (Execution Equivalence Of Process Schemes) *Let $S^{(1)}$ and $S^{(2)}$ be two process schemes. $S^{(1)}$ and $S^{(2)}$ are equivalent with respect to their execution (formally: $S^{(1)} \equiv_{execution} S^{(2)}$) iff each producible execution history $H_S^{(1)}$ on $S^{(1)}$ can be generated on $S^{(2)}$ as well and vice versa.*

Based on Def. 1 we now define the notion of equivalent changes:

Definition 2 (Equivalence Of Changes) *Let S be a process schema and Δ_1 and Δ_2 be two changes on S. Δ_i transforms S into another process schema $S^{(i)}$ $(i = 1, 2)$. Then Δ_1 and Δ_2*

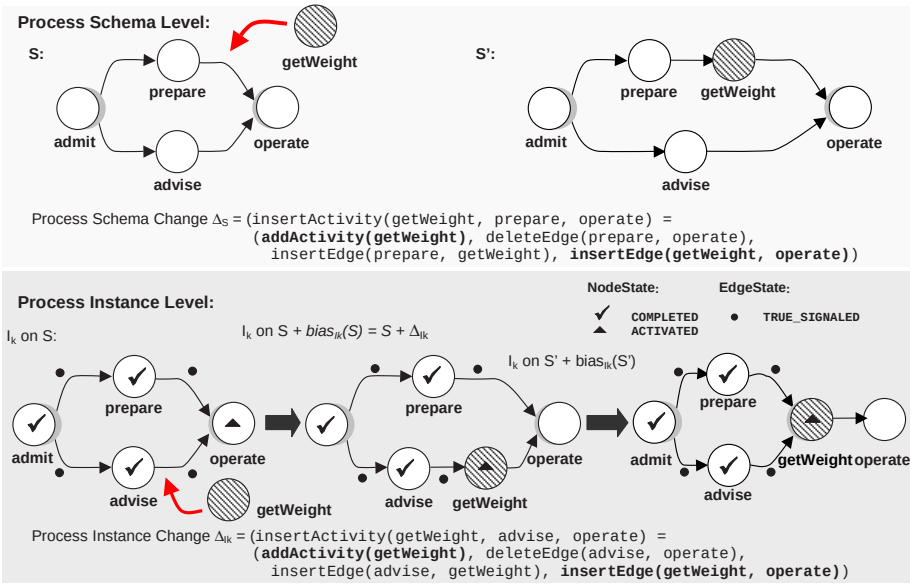


Figure 8: *Partially Equivalent Changes: Inserting An Activity In A Different Context*

are equivalent if $S^{(1)}$ *and* $S^{(2)}$ *are equivalent with respect to their execution as specified in Def. 1. Formally:*

$$\Delta_1 \equiv \Delta_2 \iff S^{(1)} \equiv_{execution} S^{(2)}$$

The important question is how to ensure execution equivalence of two process schemes $S^{(1)}$ and $S^{(2)}$. Obviously, trying to replay all possible execution histories of process schema $S^{(1)}$ on process schema $S^{(2)}$ and vice versa is far too expensive. Note that the determination of all possible execution histories of a process schema would result in exponential complexity. When surveying this question it caught our eyes that – in general – the property of execution equivalence can be transferred to the problem of **graph isomorphism (GI)** [17] between two process schema graphs $S^{(1)}$ and $S^{(2)}$, i.e., finding a bijective mapping between the nodes and edges of two graphs. A formal definition of this property is provided in Definition 3. Note that this definition is especially tailored for process schema graphs, i.e., the bijective mapping between the two process schema graphs is already fixed by the labelings of the respective schema nodes and edges.

Definition 3 (Graph Isomorphism) *Let* $S^{(i)} = (N^{(i)}, E^{(i)}, D^{(i)}, DataE^{(i)})$ *(*$i = 1, 2$*) be two process schema graphs with node set* $N^{(i)}$*, control edge set* $E^{(i)}$*, data element set* $D^{(i)}$ *and data*

edge set $DataE^{(i)} \subseteq N^{(i)} \times D^{(i)} \times \{read\_access, write\_access\}$[2]. *Then* $S^{(1)}$ *and* $S^{(2)}$ *are isomorphic (formally:* $S^{(1)} \simeq S^{(2)}$*) if condition (♣) holds with*

(♣):

$[[\exists$ *bijective mapping f:* $N^{(1)} \mapsto N^{(2)}$ *with*

$(label(n) = label(f(n)) \; \forall \; n \in N^{(1)}) \wedge$

$(\forall \; e = (u, v) \in E^{(1)}: \exists \; e^* = (f(u), f(v)) \in E^{(2)}$ *with* $label(e) = label(e^*)$

$\wedge \; \forall \; e^* = (u^*, v^*) \in E^{(2)} \; \exists \; e = (f^{-1}(u^*), f^{-1}(v^*))$

with $label(e^*) = label(e))] \wedge$

$[\exists$ *bijective mapping g:* $D^{(1)} \mapsto D^{(2)}$ *with*

$(label(d) = label(g(d)) \; \forall \; d \in D^{(1)}) \wedge$

$(\forall \; dE = (d, n, mode) \in DataE^{(1)}, \; n \in N^{(1)}:$

$\exists \; dE^{(2)} = (g(d), g(n), mode)) \in DataE^{(2)}: label(dE) = label(dE^*)$

$\wedge \; \forall \; dE^* = (d^*, n^*, mode) \in DataE^{(2)}$

$\exists \; dE = (g^{-1}(d^*), g^{-1}(n^*), mode): label(dE^*) = label(dE)]]$

The following theorem formally states that isomorphism between process schema graphs implies their execution equivalence. From this, together with Def. 2, we can derive an important perception about the relation between equivalence of changes and isomorphism of the underlying process schema graphs.

Theorem 1 (Equivalence Of Changes And Schema Graph Isomorphism) *Let S be a process schema and* Δ_1 *and* Δ_2 *two change operations.* Δ_i *transforms S into* $S^{(i)}$ $(i = 1, 2)$*. Then* $S^{(1)}$ *and* $S^{(2)}$ *are execution equivalent if* $S^{(1)}$ *and* $S^{(2)}$ *are isomorphic according to Def. 3 (*$S^{(1)} \simeq S^{(2)}$*). Furthermore, when applying Def. 2 we get that* Δ_1 *and* Δ_2 *are equivalent if* $S^{(1)}$ *and* $S^{(2)}$ *are isomorphic. Formally:*

$$S^{(1)} \simeq S^{(2)} \Longrightarrow S^{(1)} \equiv_{execution} S^{(2)} \Longleftrightarrow \Delta_1 \equiv \Delta_2$$

[2]This set-based definition of process schema graphs is common. At this point, we abstract from unnecessary details in order to not overwhelm the reader.

A formal proof can be found in the appendix. As an example for Theorem 1 take Fig. 1. Here the optimized process schema S' (schema level) and the biased execution schema S + $bias_{I_1}(S)$ (instance level) are isomorphic. One can claim is that, in general, there is no efficient algorithm for detecting graph isomorphism (GI). In particular, no efficient algorithm has been found for GI (GI lies in class NP) [17]. But there are some special graph classes for which efficient isomorphism algorithms exist, e.g., planar graphs or graphs with bounded valence [4, 20]. As pointed out in [17], however, for two graphs with unique vertex labeling it is trivial to find an isomorphim between them. As it can be seen from Def. 3 this is the case for two process schema graphs since there is always a unique labeling which is preserved during the isomorphism mapping. One way would be to compare the adjacency matrices of both process schemes. Generally, this can be done with complexity $O(n^2)$ if n is the number of nodes. Note that for arbitrary graphs one has to analyze all $n!$ permutations of the adjacency matrix. Regarding process schemes the number of rows in the adjacency matrices depends on the number of activity nodes and the number of data elements, i.e., $|N|+|D|$. Therefore the complexity of this method results in $O((|N|+|D|)^2)$

In order to to avoid accidents when comparing two process schema graphs we have to code the adjacency matrix entries with the respective edge labeling, e.g., "c" for control egdes, "tc_i" for an edge with transition condition tc_i, etc. Otherwise, such an edge attribute could be changed and anyway, the process schema graphs would be determined as isomorphic.

To our best knowledge there is no approach with lower complexity than $O(|M|^2)$ (with $M = max(|N|, |D|)$) for this special problem. This could cause a performance problem if we have to check process graph isomorphism for complex process schemes and for a large number of process instances at runtime. Nevertheless, analyzing graph isomorphism in conjunction with equivalent changes has yielded a formal foundation for the following considerations made for equivalent, subsumption equivalent, partially equivalent, and different changes (cf. Table 1).

4.2 Comparing Changes

Let S be a process schema and Δ_1 and Δ_2 be two changes on S. Δ_i transforms S into $S^{(i)}$ $(i = 1, 2)$. Intuitively, comparing $S^{(1)}$ and $S^{(2)}$ is only suitable if Δ_1 and Δ_2 are equivalent. As opposed to this, directly comparing Δ_i $(i = 1, 2)$ can be applied if the respective changes are equivalent, subsumption equivalent, partially equivalent, or different (cf. Table 1). Thereby, we assume that Δ_1 and Δ_2 can be mapped onto well-defined sets of change primitives $\Delta_i = \{p_1^{(\Delta_i)}, \ldots, p_n^{(\Delta_i)}\}$. Examples for such primitives include `deleteEdge, insertEdge`, and `changeEdgeLabel`[3]. Theorem 2 summarizes the conditions for detecting the different kinds of change relationships (cf. Table 1). They are based on the comparison of the change primitive sets related to Δ_i $(i = 1, 2)$. Thereby not only the type of two change primitives has to match but also their parametrization.

[3]For example, in ADEPT [21] change operations are defined by sets of change primitives whereby the semantics of the respective change operation is clearly determined.

Theorem 2 (Checking Change Equivalence Using Change Primitives) *Let S be a process schema and $\Delta_1 = \{p_1^{(\Delta_1)}, \ldots, p_n^{(\Delta_1)})\}$ and $\Delta_2 = \{p_1^{(\Delta_2)}, \ldots, p_m^{(\Delta_2)})\}$ be two changes on S. Δ_i transforms S into $S^{(i)}$. Then:*

(1) $\Delta_1 \equiv \Delta_2 \Longleftrightarrow \{p_1^{(\Delta_1)}, \ldots, p_n^{(\Delta_1)})\} = \{p_1^{(\Delta_2)}, \ldots, p_m^{(\Delta_2)})\}$

(2) $\Delta_1 \prec \Delta_2 \Longleftrightarrow \{p_1^{(\Delta_1)}, \ldots, p_n^{(\Delta_1)})\} \subset \{p_1^{(\Delta_2)}, \ldots, p_m^{(\Delta_2)})\}$

(3) $\Delta_1 \cap \Delta_2 \neq \emptyset \Longleftrightarrow \{p_1^{(\Delta_1)}, \ldots, p_n^{(\Delta_1)})\} \cap \{p_1^{(\Delta_2)}, \ldots, p_m^{(\Delta_2)})\} \neq \emptyset$

(4) Δ_1 *and* Δ_2 *different* $\Longleftrightarrow \{p_1^{(\Delta_1)}, \ldots, p_n^{(\Delta_1)})\} \cap \{p_1^{(\Delta_2)}, \ldots, p_m^{(\Delta_2)})\} = \emptyset$

The classification given in Theorem 2 exactly aligns with the classification provided in Table 1. We omit a formal proof and present an example instead. Exemplarily for equivalent changes ($\Delta_1 \equiv \Delta_2$) take Fig. 1. Here the set of change primitives of schema change $\Delta_S = \{$`deleteEdge(getOrder, makeInvoice), ...`$\}$ exactly aligns with the set of change primitives of instance change $\Delta_I = \{$`deleteEdge(getOrder, makeInvoice), ...`$\}$. Applying Theorem 2 we can conclude that Δ_S and Δ_I are equivalent. Fig. 8 shows an example for partially equivalent changes ($\Delta_1 \cap \Delta_2 \neq \emptyset$). The intersect of the change primitive sets of Δ_S and Δ_I results in set {`addActivity(getWeight), insertEdge(getWeight, operate)`}. Therefore Δ_S and Δ_I are partially equivalent, but they are not subsumption equivalent since Δ_S as well as Δ_I contain further change primitives (e.g., `insertEdge(prepare, getWeight)` for Δ_S and `insertEgde(advise, getWeight)` for Δ_I).

5 Treatment of Semantically Conflicting Changes

In Section 4 we have described formal methods for detecting particular relations between process schema and instance changes according to Table 1. In doing so we are able to detect possible semantical conflicts between process schema and instance changes before they cause trouble. The only thing still missing now is to provide strategies for the adequate treatment of conflicting process schema and instance changes based on their particular relation. In Section 5.1, we start with equivalent and subsumption equivalent changes since they enable semi-automatic or automatic support of the modeler / designer when propagating schema changes to the instance level. In Section 5.2, we provide a more general discussion of strategies for equivalent, subsumption equivalent, partially equivalent, and different changes whereas for partially equivalent and different changes user interaction may be required.

Let S be a process schema and let $\Delta_S = \{p_1^{(\Delta_S)}, \ldots, p_m^{(\Delta_S)}\}$ be a change which transforms S into another schema S' at time t_S. Let further $I_1, \ldots, I_n$ be a collection of biased pro-

cess instances derived from S with respective biases $bias_{I_1}, \ldots, bias_{I_n}$ at time t_S. To simplify matters, we only consider one arbitrary biased instance $I_k \in \{I_1, \ldots, I_n\}$ with bias $bias_{I_k} = \{p_1^{(bias_{I_k}(S))}, \ldots, p_l^{(bias_{I_k}(S))}\}$ in the following.

5.1 Realization Strategies For (Subsumption) Equivalent Changes

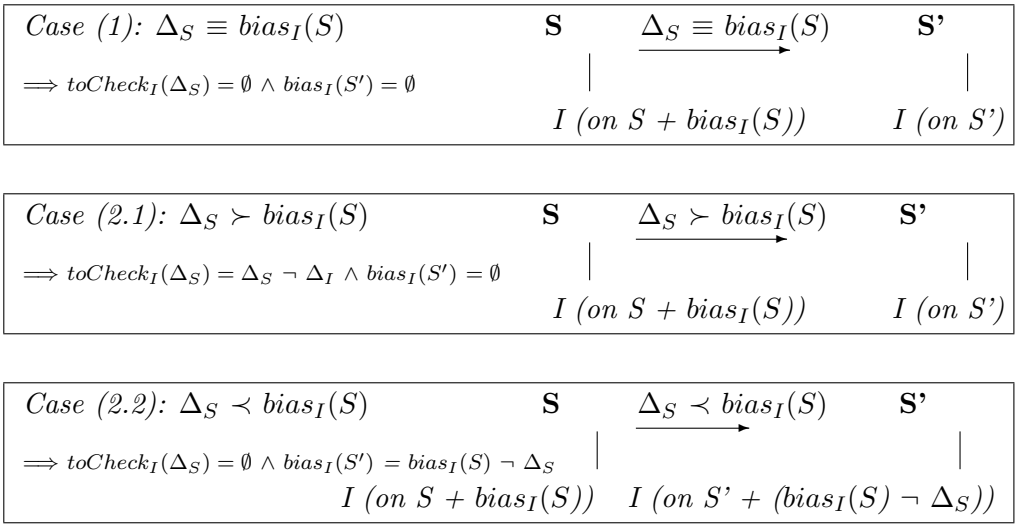
If Δ_S and $bias_{I_k}(S)$ are (subsumption) equivalent, an undesired system behavior may arise from the repeated propagation of Δ_S to I_k. Note that I_k has already anticipated this change or at least parts of it. Potential problems range from excluding I_k from migrating to S up to multiple insertion of the same activity (see Fig. 1 and Fig. 4 for examples). Therefore, the propagation of those parts of Δ_S which lie in the intersect of $\{p_1^{(\Delta_S)}, \ldots, p_m^{(\Delta_S)}\}$ and $\{p_1^{(bias_{I_k}(S))}, \ldots, p_l^{(bias_{I_k}(S))}\}$ may have to be avoided. In most cases only those parts of Δ_S shall be propagated to I_k (and therefore have to undergo corresponding compliance checks) which have not yet been included in $bias_{I_k}(S)$. Therefore we determine that subset of $\Delta_S = \{p_1^{(\Delta_S)}, \ldots, p_m^{(\Delta_S)}\}$ for which the presence of structural and state-related conflicts has to be checked when propagating Δ_S to I_k, formally:

$$toCheck_{I_k}(\Delta_S) = \{p_1^{(\Delta_S)}, \ldots, p_m^{(\Delta_S)}\} \neg \{p_1^{(bias_{I_k}(S))}, \ldots, p_l^{(bias_{I_k}(S))}\}.$$

Trivially, this set is empty in case of equivalent changes.

The other important question refers to the bias resulting after re-linking I_k to S', i.e., $bias_{I_k}(S') \subseteq \{p_1^{(bias_{I_k}(S))}, \ldots, p_l^{(bias_{I_k}(S))}\}$. As mentioned in [22], it is necessary to re-link I_k to S'; otherwise subsequent changes of S' would have no influence on I_k any longer. Theorem 3 makes a central contribution for the treatment of (subsumption) equivalent changes and their semantic conflicts: For (subsumption) equivalent changes it provides the set of change operations for which structural and state compliance checks are necessary ($toCheck_{I_k}(\Delta_S)$). Additionally, it determines the resulting bias of I_k on S' ($bias_{I_k}(S')$).

Theorem 3 (Realization of (Partially) Equivalent Changes) *Let S be a process schema and let $\Delta_S = \{p_1^{(\Delta_S)}, \ldots, p_m^{(\Delta_S)}\}$ be a change which transforms S into another process schema S' at time t_S. Let further I be a biased process instance derived from S with $bias_{I_k}(S) = \{p_1^{(bias_{I_k}(S))}, \ldots, p_l^{(bias_{I_k}(S))}\}$ at time t_S. Then I can correctly migrate to S' if structural and state-related conflicts can be counted out for $toCheck_I(\Delta_S)$. The bias resulting when re-linking I to S' is $bias_I(S')$. Thereby $toCheck_I(\Delta_S)$ and $bias_I(S')$ can be determined as follows:*



We omit a formal proof an present examples instead. Of particular interest are those cases for which we obtain $bias_I(S') = \emptyset$, i.e., Case (1) with $\Delta_S \equiv bias_I(S)$ and Case (2.2) with $\Delta_S \prec bias_I$. Note that for these change classes an immediate re-linking to the new version can be carried out without requiring further compliance checks. One example is depicted in Fig. 1 where Δ_S, $bias_{I_1}(S)$, and $bias_{I_2}(S)$ are equivalent. Thus, I_1 and I_2 can be immediately re-linked to S' without any need for additional compliance checks. Another example is depicted in Fig. 6 where $\Delta_S \prec bias_{I_k}$ applies. Here, I_k can be re-linked to S' as well without further compliance checks. The resulting bias of I_k on S' is $bias_{I_k}(S') = bias_I(S) \neg \Delta_S = \{$`deleteEdge(getOrder, makeInvoice), deleteEdge(makeInvoice, sendInvoice), ...`$\}$.

5.2 General Realization Strategies

Unfortunately, partially equivalent changes (cf. Table 1) cannot always be treated in the same way as (subsumption) equivalent changes (see Theorem 3). Propagating only $\Delta_S \neg bias_I(S)$ (taking the necessary compliance checks into account) does not always lead to the desired result (cf. Fig. 7).

Additionally, to detect semantical incompatibilities for different process schema and instance changes (cf. Table 1) we have to climb a semantical higher level. Firstly, we have to formalize the semantics of changes. Secondly, we have to state a criterion to decide whether two changes are compliant or not regarding their semantics. Here, approaches from the area of inter-workflow dependencies and semantical interoperability between workflows could be very interesting [5]. The same applies to planning techniques from the field of artificial intelligence [16]. Finally, for the formalization of the semantics of changes, ontologies [28] could be very useful. Due to lack of space we obstain from futher details here.

Table 2: *Semantically Conflicting Changes And Conflict Resolution Strategies*

$\Delta_S, bias_I(S)$ **are**	*(1) equivalent*	*(2) partially equivalent*	*(3) similar*	*(4) different*
Strategy	$\Delta_S \equiv bias_I(S)$	(2.1) $bias_I(S) \prec \Delta_S$ (2.2) $bias_I(S) \succ \Delta_S$	$\Delta_S \cap bias_I(S) \neq \emptyset$	$\Delta_S \cap bias_I(S) = \emptyset$
(A) try to migrate without checking equivalence	• unnecessary exclusion of biased instances from migating to changed schema • multiple execution of tasks			semantically undesired
(B) migration under optimized equivalence checking (Theo. 3)	automatic re-linking of biased instances	• optimized compliance checks • automatic determination of $bias_I(S')$ after migration	special cases: • multiple working of tasks (Fig. 7) • special structure of S' + $bias_I(S')$	not possible
(C) rolling back $bias_I(S)$	depends on instance state but implicitly contained in optimized method (cf. Theo. 3)	(2.1) same as for $\Delta_S \equiv bias_I(S)$ (2.2) only correct if I is partially rolled back s. t. $bias_I(S) \neg \Delta_S$ is maintained	partial or complete roll-back of $bias_I(S)$ could be desired if optimized method (Theo. 3) leads to multiple task execution (Fig. 7)	possible but loss of information
(D) user defined action may be conceivable as well (e.g., Fig. 8)				

Consequently, for partially equivalent as well as for different (but semantically non-compliant) changes we have to search for other strategies than those described in Theorem 3. Of course, also for (subsumption) equivalent process schema and instance changes alternative realization methods are conceivable. Possible strategies are summarized in Table 2.

6 Summary and Outlook

In this paper, we have elaborated fundamental issues related to semantically conflicting process schema and process instance changes. For the first time, we have classified semantical conflicts by distinguishing between equivalent, subsumption equivalent, partially equivalent, and different changes leading to these conflicts. We believe that the adequate treatment of semantically conflicting changes is crucial for the intelligent support of process adaptations in advanced application environments (with many users and hundreds up to thousands of long-running process process instances). In order to enable the process management system to detect semantically conflicting changes we have provided two different formal methods – one based on execution equivalence of the respective process schemes, the other based on a direct comparison of the applied changes. Additionally, we have introduced sophisticated strategies to adequately deal with semantical conflicts. Altogether, the presented concepts constitute a good basis for the adequate treatment of semantically conflicting changes and will form a key part of process flexibility in future adaptive process management software. Finally, we have implemented a powerful proof-

of-concept prototype for dynamic process changes which is presently on its way to incorporate semantical issues as well.

The considerations made in this paper additionally show that one cannot treat the different kinds of dynamic process changes occuring in practice in an isolated fashion only. That means adaptive process technolgy must enable both, ad-hoc changes of individual process instances as well as process schema changes, and it must provide a common framework for this. Unfortunately, in most suggestions made in the workflow literature so far, once a process instance has been individually modified it may no longer take part at future migrations to new process schema versions. However, this restriction totally conflicts with the needs of real-world processes. In order to adequately consider such requirement, from the very beginning, our work has been accompanied by projects dealing with real-world processes (particularly from the clinical and the engineering domain) [8, 7, 3].

Concerning adaptive process technology, we strongly believe that semantic issues constitute a field that would benefit by a more intense study of the research community. Generally, by incorporating more semantics into the design, implementation, and modification of process-oriented applications, it will become possible to support process changes at a higher semantic level and in a more comprehensible and user-friendly way as in current adaptive process management software. In future, we will extend our work on semantical aspects. Among other things we will address semantic issues related to the process-oriented composition of application services (e.g., web services) in a plug-and-play style. In this context, the use of domain-specific ontologies seems to be very promising. The incorporation of ontologies may also facilitate the definition of ad-hoc changes by end users, for example concerning the dynamic plug-in (insertion) of application components and the interconnection of their input/output parameters with process data elements. Finally, we believe that adaptive process technologies would also benefit from the incorporation of planning techniques from the field of artifical intelligence.

References

[1] A. Agostini and G. De Michelis. Improving flexibility of workflow management systems. In *Proc. Int'l Conf. BPM'00*, LNCS 1806, pages 218–234, 2000.

[2] V. Atluri, W. Huang, and E. Bertino. A semantic based execution model for multilevel secure workflows. *Int'l Journal of Computer Security*, 8(1):3–41, 2000.

[3] T. Beuter, P. Dadam, and P. Schneider. The WEP model: Adequate workflow-management for engineering processes. In *Proc. Europ. Conf. on Concurrent Engineering*, Erlangen, April 1998.

[4] H. Bunke and X. Jiang. Graph matching and similarity. In H. N. Teodorescu, D. Mlynek, A. Kandel, and H. J. Zimmermann, editors, *Proc. Intelligent Systems and Interfaces*. Kluwer Academic Publishers, 2000.

[5] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Semantic workflow interoperability. In *Proc. Int'l Conf. EDBT '96*, LNCS 1057, pages 443–462, Avignon, March 1996. Springer.

[6] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.

[7] P. Dadam and M. Reichert. Towards a new dimension in clinical information processing. In *Proc. MIE2000/GMDS 2000*, pages 295–301, Hannover, Sept. 2000.

[8] P. Dadam, M. Reichert, and K. Kuhn. Clinical workflows – the killer application for process-oriented information systems? In *Proc. 4th Int'l Conf. on Business Information Systems (BIS '00)*, pages 36–59, Poznan, Poland, 2000.

[9] D. Edmond and A.H.M. ter Hofstede. A reflective infrastructure for workflow adaptability. *Data and Knowledge Engineering*, 34(3):271–304, 2000.

[10] C. Ellis and K. Keddara. A workflow change is a workflow. In *Proc. Int'l Conf. BPM'00*, volume 1806 of *LNCS*, pages 516–534. Springer, 2000.

[11] C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proc. Int'l Conf. on Org. Comp. Sys.*, pages 10–21, Milpitas, August 1995.

[12] A. Fent, H. Reiter, and B. Freitag. Design for change: Evolving workflow specifications in ULTRAflow. In *Proc. Int'l Conf. CAISE'02*, pages 516–534, Toronto, May 2002.

[13] H. Frank and J. Eder. Equivalence transformations on statecharts. In *Proc. Int'l Conf. on Softw. Engineering. and Knowledge Engineering.*, pages 150–158, Chicago, July 2000.

[14] J. Herbst and D. Karagiannis. Intergrating machine learning and workflow management to support acquisition and adaption of workflow models. In *Proc. 9th Int'l DEXA'98 Workshop*, pages 745–752, Vienna, August 1998.

[15] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *Proc. Int'l Conf. CoopIS'98*, pages 310–321, New York City, August 1998.

[16] M. Klein, C. Dellaroca, and A. Bernstein. Towards adaptive workflow systems. In *Int'l CSCW-98 Workshop*, Seattle, Nov. 1998.

[17] J. Köbler, U. Schöning, and J. Toran. *The Graph Isomorphism Problem: Its Structural Complexity.* Birkhäuser Verlag, Boston, 1993.

[18] K. Kochut, J. Arnold, A. Sheth, J. Miller, E. Kraemer, B. Arpinar, and J. Cardoso. IntelliGEN: A distributed workflow system for discovering protein-protein interactions. *Distributed and Parallel Databases*, 13:43–72, 2003.

[19] M. Kradolfer and A. Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Proc. CoopIS '99*, pages 104–114, Edinburgh, September 1999.

[20] E. Luks. Isomorphism of graphs of bounded valance can be tested in polynomial time. *Journal of Computer und System Science*, 25(1):42–65, 1982.

[21] M. Reichert and P. Dadam. ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Inf. Syst.*, 10(2):93–129, 1998.

[22] M. Reichert, S. Rinderle, and P. Dadam. A formal framework for workflow type and instance changes under correctness constraints. Technical Report UIB-2003-01, University of Ulm, Computer Science Faculty, April 2003.

[23] S. Rinderle, M. Reichert, and P. Dadam. Evaluation of correctness criteria for dynamic workflow changes. In *Proc. Int'l Conf. on Business Process Management (BPM '03)*, LNCS 2678, pages 41–57, Eindhoven, June 2003.

[24] C. Rolland. A comprehensive view of process engineering. In *Proc. Int'l Conf. CAiSE '98*, LNCS 1413, pages 1–24, Pisa, June 1998. Springer.

[25] S. Sadiq, O. Marjanovic, and M. Orlowska. Managing change and time in dynamic workflow processes. *Int'l Journal of Coop. Inf. Syst.*, 9(1&2):93–116, 2000.

[26] S. Sadiq and M. Orlowska. Dynamic modification of workflows. Technical Report 442, Department of Computer Science and Electrical Engineering, University of Queensland, Brisbane, Australia, October 1998.

[27] S. Sadiq and M. Orlowska. Architectural considerations in systems supporting dynamic workflow modification. In *Proc. Workshop Software Architectures for Business Process Management*, Heidelberg, June 1999.

[28] R. Studer, V.R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Int'l Journal of Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[29] W.M.P. van der Aalst. Exterminating the dynamic change bug: A concrete approach to support worfklow change. *Information Systems Frontiers*, 3(3):297–317, 2001.

[30] W.M.P van der Aalst and T. Basten. Inheritance of workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[31] J. Wäsch and W. Klas. History merging as a mechanism for concurrency control in cooperative environments. In *Proc. RIDE'96 Workshop*, pages 76–85, New Orleans, Feb. 1996.

[32] M. Weske. Flexible modeling and execution of workflow activities. In *Proc. 31st Int'l Conf. on System Sciences*, pages 713–722, Hawaii, 1998.

[33] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *Proc. 34th Hawaii Int'l Conf. on System Sciences (HICSS-34)*, Los Alamitos, 2001.

Proof: Theorem 1:

In the following, for process schema S let Ω_S comprise all producable execution histories on S.

Proposition: $\mathrm{S}^{(1)} \simeq \mathrm{S}^{(2)} \Longrightarrow \mathrm{S}^{(1)} \equiv_{execution} \mathrm{S}^{(2)}$

We show this by induction over the length k of an arbitrary execution history $H_{S^{(i)}} =< e_1, \ldots, e_k > \in \Omega_{S^{(i)}}$ (i = 1, 2) with $e_j \in \{\texttt{START}(\text{label(a)}), \texttt{END}(\text{label(a)})\}$; $j = 1, \ldots, k$; a $\in N^{(i)}$.

Inductive Assumption (IA):
$\mathrm{S}^{(1)} \simeq \mathrm{S}^{(2)} \Longrightarrow \forall\ H_{S^{(1)}} =< e_1, ...e_k > \in \Omega_{S^{(1)}}$: $H_{S^{(1)}} \in \Omega_{S^{(2)}}$,
i.e., $H_{S^{(1)}}$ can be produced on $\mathrm{S}^{(2)}$ as well.

Note that we restrict our considerations to the direction from $\mathrm{S}^{(1)}$ to $\mathrm{S}^{(2)}$. Trivially, the reverse direction – $\forall\ H_{S^{(2)}} =< e_1, e_2, ...e_k > \in \Omega_{S^{(2)}}$: $H_{S^{(2)}} \in \Omega_{S^{(1)}}$ – can be proven analogously.

Inductive Beginning (IB):
$H_{S^{(1)}} =< e_1 > \in \Omega_{S^{(1)}}$ with $e_1 = \texttt{START}(\textit{label}(\text{a}))$

With condition (♣) from Def. 3 it follows that there is an image activity f(a) $\in N^{(2)}$ ($N^{(2)}$ denotes the node set of $\mathrm{S}^{(2)}$) with *label*(f(a)) = *label*(a). Since activity "a" has written the first entry into $H_{S^{(1)}}$ it must be a start activity of $S^{(1)}$, i.e., a node without incoming control edges. With (♣) from Def. 3 it directly follows that the image f(a) $\in N^{(2)}$ in $\mathrm{S}^{(2)}$ has no incoming control edges as well (edge order preserving property). Consequently, activity execution order given by $H_{S^{(1)}}$ can be produced on $\mathrm{S}^{(2)}$ as well.

However, we still have to care about the possible read accesses produced by activity "a". Due to $e_1 = \texttt{START}(\textit{label}(\text{a}))$ activity "a" was already started, i.e., it has already read the set of process data elements $D_a^{(1)} \subseteq D^{(1)}$ to which it is linked via a set of read data edges $DataE_a^{(1)} \subseteq DataE^{(1)}$. With (♣) from Def. 3 it follows: $\forall d_j \in D_a^{(1)}$: $\exists\ g(d_j) \in D_a^{(2)}$ with label(d_j) = label($g(d_j)$) $\wedge$ $\forall dE_i \in DataE_a^{(1)}$: $\exists\ g(dE_i) \in DataE_a^{(2)}$ with label(dE_i) = label($g(dE_i)$). Consequently, all entries produced by read data accesses of a in $H_{S^{(1)}}$ can be produced on $\mathrm{S}^{(2)}$ as well.

Inductive Step (IS): $H_{S^{(1)}} =< e_1, e_2, ..., e_k, e_{k+1} > \in \Omega_{S^{(1)}}$

Let $a \in N^{(1)}$ be the activity which has written entry e_k into $H_{S^{(1)}}$ and let $\tilde{a} \in N^{(1)}$ be the activity which has written e_{k+1} into $H_{S^{(1)}}$.

Due to **(IA)** it follows that $H'_{S^{(1)}} < e_1, e_2, ..., e_k > \in \Omega_{S^{(2)}}$. Now we analyze the order relation beween activities a and $\tilde{a}$ regarding $S^{(1)}$.

Case 1: $a = \tilde{a} \Longrightarrow e_k = \texttt{START}(\text{a}) \wedge e_{k+1} = \texttt{END}(\text{a})$ (*)

Taking **(IA)** and (*), trivially, $H_{S^{(1)}} \in \Omega_{S^{(2)}}$ holds.

Case 2: $a \neq \tilde{a}$

For this case, activity a is either a direct predecessor of $\tilde{a}$ or a and $\tilde{a}$ are ordered in parallel regarding schema $S^{(1)}$.

Case 2.1: a is a direct predecessor of $\tilde{a}$

With (♣) from Def. 3 it follows:
$\exists$ f(a), f($\tilde{a}$) in $N^{(2)}$ with: f(a) is a direct predecessor of f($\tilde{a}$) . For this case, it directly follows that $H_{S^{(1)}} \in \Omega_{S^{(2)}}$ holds.

Case 2.2: a and $\tilde{a}$ are ordered in parallel

With (♣) from Def. 3 it follows:
$\exists$ f(a), f($\tilde{a}$) in $N^{(2)}$ with: f(a) and f($\tilde{a}$) are ordered in parallel . For this case, trivially, $H_{S^{(1)}}$ $\in \Omega_{S^{(2)}}$ holds.

However, we still have to care about the possible read and write data accesses produced by activity $\tilde{a}$. In doing so, we distinguish between the following cases:

Case 1: $e_{k+1} = \texttt{START}(\text{label}(\tilde{a}))$

Activity $\tilde{a}$ has read the set of process data elements $D_{\tilde{a}}^{(1)} \subseteq D^{(1)}$ to which it is linked via a set of read data edges $DataE_{\tilde{a}}^{(1)} \subseteq DataE^{(1)}$. With (♣) from Def. 3 it follows:
$\forall d_j \in D_{\tilde{a}}^{(1)}$: $\exists\ g(d_j) \in D_{\tilde{a}}^{(2)}$ with label($g(d_j)$) = label(d_j)
$\wedge\ \forall dE_i \in DataE_{\tilde{a}}^{(1)}$: $\exists\ g(dE_i) \in DataE_{\tilde{a}}^{(2)}$ with label($g(dE_i)$) = label(dE_i)

Case 2: $e_{k+1} = \texttt{END}(\text{label}(\tilde{a}))$

Since $\tilde{a}$ has been already completed, there were write data accesses of $\tilde{a}$ on a set of data elements $D_{\tilde{a}}^{(1)} \subseteq D^{(1)}$. For them we can argue analogously to read data accesses in Case 1.