# Advanced Migration Strategies for Adaptive Process Management Systems

Stefanie Rinderle-Ma
University of Vienna, Austria
Faculty of Computer Science
stefanie.rinderle-ma@univie.ac.at

Manfred Reichert
Ulm University, Germany
DBIS Institute
manfred.reichert@uni-ulm.de

## Abstract

*Enabling dynamic process changes is an essential requirement for any adaptive process management technology. Particularly, it should be possible to migrate (long-) running process instances to a new process schema version. Further, instance migration must not violate soundness; i.e., structural and behavioral consistency of executed process schemas need to be preserved. State compliance has been introduced as basic correctness notion to ensure that instances, whose state has progressed too far, are prohibited from being migrated. However, this also excludes them from future process optimizations, which is often not tolerable in practice. This paper introduces advanced migration strategies for coping with non-compliant instances in the context of process change such that they can benefit from future process type changes on the one hand, but do not run into soundness problems on the other hand. Contrary to existing approaches, the strategies proposed in this paper are based on adjustments at process type and instance level. Altogether, the suggested migration strategies complement treatment of non-compliant process instances.*

## 1 Introduction

The ability to effectively deal with change has been identified as key functionality for any process-aware information systems (PAIS). Through the separation of process logic from application code, PAIS facilitate process changes significantly. In the context of long-running processes (e.g., medical treatment), PAIS should additionally enable the propagation of such process changes to ongoing process instances. Regarding the support of dynamic process changes, PAIS robustness is fundamental; i.e., changes must not violate soundness [14] of the running process instances. In response to this challenge adaptive process management technology has emerged, which enables dynamic process changes at different levels [1, 8, 11, 12, 14]. Most approaches apply a specific correctness notion to ensure that only those process instances may migrate to a modified process schema for which soundness is maintained afterwards. One of the most prominent criteria used in this context is *state compliance* [1, 8]. Based on state compliance it is ensured that instances, whose state has progressed too far, are prohibited from being migrated. However, taking the experience we gathered when applying our adaptive process management system AristaFlow (www.aristaflow-forum.de) in practice we learned that state compliance is sometimes too restrictive in respect to the migration of long-running processes (see next paragraph).

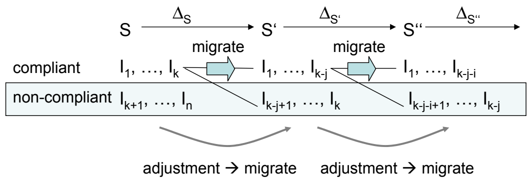


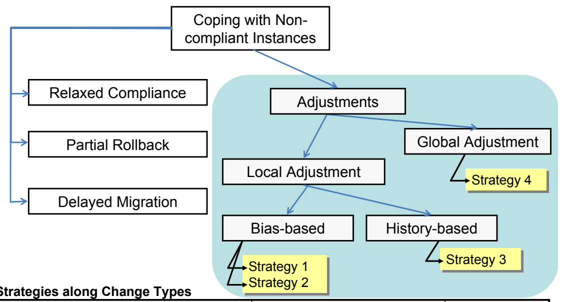
**Figure 1. Instance Migration**

Consider the abstract scenario from Fig. 1: On process schema $S$, instances $I_1, ..., I_n$ are running. Then $S$ is transformed into new schema version $S'$ by applying process change $\Delta_S$. Assume that it can be decided that $I_1, ..., I_k$ may migrate to $S'$ without violating soundness (i.e., they are compliant with $S'$), whereas $I_{k+1}, ..., I_n$ have already progressed too far (i.e., they are not compliant with $S'$).[1] As suggested by most approaches $I_{k+1}, ..., I_n$ then remain running on $S$. Assume that a further optimization of the new schema version $S'$ takes place captured by process change $\Delta_{S'}$ resulting in new type schema version $S''$. Then $I_1, ..., I_k$ could also benefit from $\Delta_{S'}$ if they were compliant with $S''$. However, $I_{k+1}, ..., I_n$ are excluded from migration to $S''$ anyway since they are not running on $S'$ at all. This is not tolerable in many practical settings. Think of, for example, a patient who is excluded from an optimized treatment because his particular process instance can-

[1] We assume that instances are clustered along their state.

not be migrated to the changed process type schema. Or consider engineering processes that require changes due to the replacement of an already released product component. Regarding change management processes in the automotive domain, for example, we learned that it must be also possible to adjust already passed process regions without rolling back the process. Or in a project with an electricity supplier, we experienced that in the context of schema evolution, non-compliant instances were adjusted individually to re-link them to the new process schema version.

In this paper, we introduce advanced strategies for coping with non-compliant instances in the context of process change such that they can benefit from future process changes on the one hand, but do not run into soundness problems on the other hand. As depicted in Fig. 2 existing approaches treat non-compliant process instances by rolling them back [11], by migrating them with delay [3], or by enabling a higher number of process instances to migrate based on relaxing compliance notions [10]. The approach presented in this paper, can be seen as complementary to all these approaches. The decision which strategy is to applied, might depend on the application context. We will give a short discussion on this in Sect. 5. Note that our work provides the technical foundation for improving migratability of instances in adaptive PAIS. Additionally, semantical issues can become relevant for deciding on certain cases. We will refer to this issue in Sect. 5 based on our work within the SeaFlows project [5] on semantic-aware processes.



**Strategies along Change Types**

| Control Flow Changes | | | Data Flow Changes (accompanying) | ANY OTHER |
|---|---|---|---|---|
| INSERT | DELETE | MOVE | | |
| Strategy 1<br>Strategy 2<br>Strategy 4 | Strategy 1<br>Strategy 3 | Strategy 1<br>Strategies 2/4<br>+ Strategy 3 | Strategy 1<br>Strategy 2 :<br>- Single insertions: ✓<br>- Multiple insertions: cf. Sect. 5 | Strategy 1 |

**Figure 2. Treating Non-Compliant Instances**

Section 2 introduces background information. Section 3 presents different strategies for dealing with non-compliant instances. Section 4 shows how non-compliant process instances can be structurally adjusted to make them migrateable. Section 5 provides an evaluation for applying the different strategies. Section 6 discusses related work and Section 8 closes with a summary.

## 2 Background

We first summarize basic concepts and notions used in this paper. For each *business process* to be supported a *process type* T represented by a *process schema* S has to be defined. For a particular type several process schemas may exist, representing the different *versions* and *evolution* of this type over time. In the following, a single process schema is represented as directed graph, which comprises a set of nodes – representing *activities* or *control connectors* (e.g., XOR-Split, AND-Join) – and a set of *control edges* (i.e., precedence relations) between them. In the following we assume relaxed *block-structured* process schemas; i.e., on top of block-based process patterns, we provide *sync links* for setting up order relations between activities belonging to parallel branches (similar to the link concept in BPEL). In this paper, we assume acyclic process structures. In Sect. 5 we provide a discussion on how the proposed migration strategies can work in connection with loops as well.

Further, a process schema comprises data elements and data edges. A *data edge* links an activity with a *data element* and represents a read/ write access of this activity to the respective data element. Altogether, a process schema $S$ is represented by a tuple $S := (N, CtrlE, SyncE, D, DataE, ...)$ where $N$ denotes the set of activities, *CtrlE* the set of control egdes, *SyncE* the set of sync edges, $D$ the set of data elements, and *DataE* the set of data edges. Based on process schema $S$ at runtime *process instances* can be created and executed. For instance $I$ running on $S$, start and/or completion events of corresponding activities are recorded in *execution trace* $\sigma_I^S$. A compact representation of trace $\sigma_I^S$ is given by instance marking $NS_I$ which assigns to each activity of $I$ its current state. Possible activity states include *NotActivated*, *Activated*, *Running*, *Completed*, and *Skipped* [8].

In general, we assume *sound* process schemas $S$ [2]. Following the definition given in [14], we distinguish between structural and behavorial soundness. Basically, a schema $S$ is structurally sound if it has exactly one start and one end activity and all activities are connected. There might be additional structural soundness constraints set out by the particular process meta model (e.g. block-structuredness). In summary, behavorial soundness refers to correct instance states [8], i.e., the instance must not run into any livelock or deadlock. Additionally, input data has to be correctly supplied for all activities. If, for example, an activity is started before all predecessor activities are completed, this activity will not necessarily be supplied with all required input data.

**Process change:** Basically, changes can be triggered and performed at the process type and process instance level. *Changes of a process type T* may become necessary to cover the evolution of the business processes captured in process schemas of this type [12, 8, 14]. By contrast, *changes of* *individual process instances* are usually performed by end users and become necessary to react to exceptional situations [8]. In particular, effects of such changes must be kept local, i.e., they must not affect other instances of same type. Thus for each individually modified instance $I$ an instance-specific schema $S_I$ is maintained. The difference between $S_I$ and original schema $S$ is captured by the so called instance-specific *bias* of I; i.e., $\Delta_I(S)$. Specifically $\Delta_I(S)$ captures all change operations applied to $S$ at instance level in order to obtain instance-specific schema $S_I$. Instances which have not been individually modified are denoted as *unbiased* instances.

In the context of process type and process instance changes, structural and behavorial soundness have to be preserved. In our ADEPT approach, structural soundness is achieved based on well-defined pre- and post-conditions for the different change operations [8]. Behavorial soundness is accomplished by behavorial correctness criteria. One prominent example is the *state compliance criterion* [1] which is used in the remainder of this paper. Informally, state compliance checks whether execution trace $\sigma_I^S$ of instance $I$ on schema $S$ could be also produced by an instance on $S'$ in the same order as set out by $\sigma_I^S$.

**Metrics on Process Schemas:** One possiblity to evaluate the migration strategies proposed in the following is to measure the difference between original process schema $S_1$ and schema $S_2$ resulting after applying a certain migration strategy. For this we apply process metrics like *activity coverage* and *process similarity*. Further we must consider the number of control relations being different for $S_1$ and $S_2$, denoted by $dS(S_1, S_2)$. Informally, for each node in $S_1$ ($S_2$) all successor nodes are counted that are different from the successor nodes of $S_2$ ($S_1$). This number should be minimized in order to enable propagation of subsequent schema changes (see Sect. 5 for definitions of these metrics).

## 3 Treating non-compliant instances

This section discusses strategies for treating non-compliant instances. Certain kinds of adjustment (cf. Fig. 2) enable non-compliant instances to migrate to the new schema version if desired by the process engineer; i.e., to relink them to the new type schema. One of the possible adjustments is to "simulate" an instance-specific bias such that the effects of the type change, for which the instance has progressed too far, are "neutralized". This enables migration or – more precisely – "re-linking" of the respective instance to the new type schema version. Consequently, this instance can benefit from further schema optimization.

**Strategy 1 (Always-Migrate):** Basically, **any** non-compliant instance can be migrated (i.e., relinked) to modified type schema version. The idea is as follows: Let $S$ be a process schema which is transformed into $S'$ by change $\Delta_S$. Let further $I$ be an instance on $S$. So far, $\Delta_S$ is propagated to $I$ when migrating $I$ to $S$ (i.e., $I$ reflects $\Delta_S$ after its migration to $S'$). However, if $I$ is not compliant with $S'$, $\Delta_S$ must not be applied to $I$; i.e., execution of $I$ should be continued on its old schema. However, at technical level this effect can be "simulated" when re-linking $I$ to $S'$ by "neutralizing" type change $\Delta_S$. "Neutralizing" means to introduce an artificial bias $\Delta_I$ on $S'$ which reverses the effects of $\Delta_S$ for this particular instance. Consider Fig. 3. At type level activity $X$ is inserted between activities $A$ and $B$ including the insertion of a read data edge from $X$ on data element $d$. Then instance $I$ (cf. Fig. 3b) is not compliant with S' since $B$ has been already completed. Nevertheless, $I$ can be migrated to $S'$ by not applying type change $\Delta_S$. The invalid change $\Delta_S$ for $I$ on $S'$ can be "healed" by creating an instance-specific bias $\Delta_I(S')$ for $I$ on $S'$; $\Delta_I(S')$ then reflects the deviations between $S'$ and instance-specific schema $S_I$. Specifically, $\Delta_I(S')$ constitutes the "inverse" change operation of $\Delta_S$. In our example, adding $X$ to $S$ can be reversed by deleting read data edge $X \rightarrow d$ as $X$ from $S'$. Strategy 1 can be applied for any kind of change regardless whether they concern control or data flow. The challenge is to determine the "inverse" change to be maintained for the migrated instance. Tab. 1 shows inverse changes for activity insertion, deletion, and order-changes as well as for selected data flow changes. Alternatively, we can directly determine schema "difference" between type schema $S'$ and instance-specific schema, e.g., [4] presents an approach for comparing two schemas $S_1$ and $S_2$ in terms of high-level change operations needed to transform $S_1$ into $S_2$.

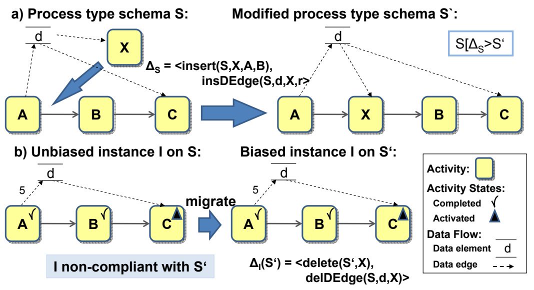

a) Process type schema S:
Modified process type schema S`:
d
X
Δ_S = <insert(S,X,A,B), insDEdge(S,d,X,r>
S[Δ_S>S'
A
B
C
A
X
B
C
b) Unbiased instance I on S:
Biased instance I on S':
5
d
migrate
A√
B√
C
I non-compliant with S'
Δ_I(S') = <delete(S',X), delDEdge(S,d,X)>
Activity:
Activity States:
Completed √
Activated
Data Flow:
Data element d
Data edge

Figure 3. Strat. 1: Always-Migrate (Example)

**Strategy 2 (Instance-Specific Adjustment)** The idea behind Strategy 2 is to conduct instance-specific adjustments of type changes. This way we are able to relink non-compliant instances to the new type schema version. For this purpose we exploit the specific semantics of the applied change operation [13]: When inserting an activity, for example, the user has to specify the position where to insert the new activity. Thus activity insertion is a candidate for instance-specific change adjustment since such position parameters can be easily adapted; e.g., by inserting the activtiy

| **Change operation $\Delta$ on S** | opType | subject | paramList | inverseOp |
|---|---|---|---|---|
| insert(S, X, A, B)[a] | insert | X | S, A, B | delete(S, X) |
| **Effects on S:** inserts activity S between activities A and B. | | | | |
| delete(S, X) | delete | X | S | insert(S, X, pred(S,X), succ(S,X))[b] |
| **Effects on S:** deletes activities from S | | | | |
| move(S, X, A, B) | move | X | S, A, B | move(S, X, pred(S,X), succ(S,X)) |
| **Effects on S:** moves activity S from its original position in S to another position between activity sets A and B | | | | |
| insDE(S, d) | insert | d | S | delDE(S,d) |
| **Effects on S:** inserts data element d into S | | | | |
| delDE(S, d) | delete | d | S | insDE(S,d) |
| **Effects on S:** deletes data element d from S (d is not referenced by any data connector) | | | | |
| insDEdge(S, d, X, [r\|w]) | insert | d | S, X, [r\|w] | delDEdge(S,d,X) |
| **Effects:** inserts read/write dada edge between activity X and data element d into S | | | | |
| delDEdge(S, d, X) | delete | d | S, X | insDEdge(S, d, X, [r\|w]) |
| **Effects:** deletes data edge between activity X and data element d | | | | |

[a] Basically, insertion between activities A and B can be generalized to insertion between activity sets $\mathcal{A}$, $\mathcal{B}$. Additionally, for conditional insert an optional parameter [sc] can be included for representing a conditional insert (for details see [13]).

[b] pred(S,X) / succ(S,X) denotes the direct predecessor(s) / successor(s) of X in S.

**Table 1.** *Selection of Change Operations on Process Schemas*

"later" than originally intended.

Consider the example depicted in Fig. 4. Instance $I$ (cf. Fig. 4b) is not compliant with new type schema $S'$ since $B$ has state `Running`. Basically, if possible from a semantic point of view, at instance level $X$ can be inserted at other positions as well such that $I$ becomes compliant again. Assuming that we keep $A$ as "left anchor" for the insert operation, possible "right anchors" for the insertion are $C$ and $D$ respectively. First consider insertion of $X$ between $A$ and $C$ (①): When applying $\Delta_I^1(S)$ to $I$ as instance-specific adjustment, we obtain instance schema $S_I^1$. The bias between $S_I^1$ and $S'$ is captured by $\Delta_I^1(S')$ and reflects an insertion with "relaxed" insertion position.[2] Schema $S_I^2$ resulting from the insertion of $X$ between $A$ and $D$ is depicted in Fig. 4b (②). Obviously, the question is which of the two schemas we shall prefer. The premise of this paper is to enable migration of non-compliant instances such that they can benefit from further optimizations. In general, the application of further modifications at type level will be supported best when keeping the instance-specific schemas "as close as possible" to the type schema version they are migrated to. Intuitively, $S_I^1$ is "closer" to $S'$ when compared to $S_I^2$. To define "as close as possible" we can use structural distance dS as introduced in Section 2.

Basically, it is also possible to use "left anchors" other than $A$ for realizing instance-specific adjustments. Consider Fig. 4b(③): When compared to $S'$ the order between $X$ and $B$ has been reversed for Fig. 4b(③), whereas for Fig. 4b(①) $X$ and $B$ are ordered in parallel. Thus in Fig. 4b(①), still $X$ can be started before $B$ is finished, what is not possible in Fig. 4b(③). Hence, the instance schema from Fig. 4b(①) is "closest" to $S'$. – Generally, we want to answer: *how to determine instance-specific adjustments $\Delta_I(S')$ such that I is compliant with $S_I$ where $S'[\Delta_I(S') > S_I$[3] and distance between instance schema $S_I$ and modified type schema $S'$ $dS(S_I, S')$ becomes minimal?*

[2] i.e., $X$ is not inserted between $A$ and $B$, but between $A$ and the first "possible" successor of $B$.

**Strategy 3: History-Based Adjustment** History-based adjustment is mainly applied for delete operations: e.g., an instance is not compliant if the activity to be deleted is running or completed. Basically, deletions of activities "in the past" of process instances are enabled by adjusting instance traces[4]. Assume that activity $X$ is to be deleted from schema $S$ resulting in schema $S'$. Regarding instance $I$ on $S$, $X$ has been already completed and respective `start and/events` for $X$ were logged in trace $\sigma_I^S$. Thus $\sigma_I^S$ cannot be "replayed" on $S'$ and compliance for $I$ on $S'$ is not fulfilled. However, when discarding respective trace entries of $X$ from $\sigma_I^S$, the modified trace can be replayed on $S'$ and thus $I$ be migrated. Entries of deleted activities are physically not deleted from the exeuction traces, but are only flagged within the trace to preserve traceability. Strategy 3 does not harm soundness of the affected instances.

**Strategy 4: Global-Adjustment** Basically, adjusting schema changes may be done at both, the process type and the process instance level. Assume that process type schema $S$ is transformed into another type schema $S'$ by change $\Delta_S$. Let further $I$ be an instance on $S$ which is not compliant with $S'$. If $\Delta_S$ is adjusted to $\Delta'_S$ at type level (i.e., transforming $S$ into $S''$ instead of $S'$), more instances running on $S$ become compliant with $S''$ afterwards. Generally, more instances will become compliant with a changed type schema, if added activities are inserted as late as possible. Most important, all data dependencies imposed by the process type schema and the intended change must be fulfilled. For the given example this implies that activities

[3] An interesting question arises in the context of data flow changes accompanying insert operations as, for example, depicted in Fig. 3

[4] For practical scenarios from the automotive domain in which such non-compliant deletions are required, we refer to [6].

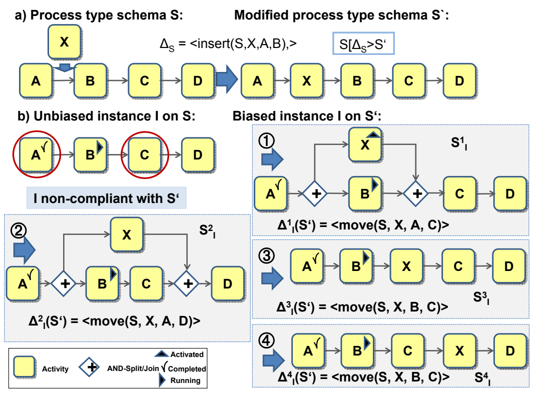

a) Process type schema S:
Modified process type schema S`:
Δ_S = <insert(S,X,A,B),>
S[Δ_S>S'
b) Unbiased instance I on S:
Biased instance I on S':
I non-compliant with S'
①
S¹_I
Δ¹_I(S') = <move(S, X, A, C)>
②
S²_I
Δ²_I(S') = <move(S, X, A, D)>
③
Δ³_I(S') = <move(S, X, B, C)>
S³_I
④
Δ⁴_I(S') = <move(S, X, B, C)>
S⁴_I
Activity
AND-Split/Join
Activated
Completed
Running

**Figure 4. Strat. 2: Instance-Specific Adjustment (Example)**

X and Y can be inserted "later in the type schema" (i.e., as "close" to the process end as possible) as long as the data dependency between them is fulfilled. Since a process type schema might contain more than one process end node, the formalization of "later in a process graph" should not be based on structural properties; i.e., we want to be independent of a particular process meta model. As for state compliance, we use process traces in this context. Due to lack of space we omit a formalization here. Finally, when inserting two or more data-dependent activities, additional constraints must hold. More precisely, it cannot be allowed to move the insertion position of the writing activity "behind" the reading activity since the resulting schema would not be correct anymore.

## 4 Instance-Specific Adjustment of Instances

Bias-based strategies as introduced in Sect. 3 are promising approaches for treating non-compliant instances. Assume that instance $I$ on $S$ is not compliant with modified type schema version $S'$ ($S[\Delta_S > S'$). However, if type change $\Delta_S$ can be locally adjusted to $\Delta_I(S')$ such that $I$ becomes compliant with instance schema $S[\Delta_I(S') > S_I$, $I$ can be relinked to $S'$ using bias $\Delta_I(S')$ (Strategy 2). As motivated by Fig. 4, different adjustments of $\Delta_S$ are conceivable. However, we are particularly interested in the adjustment which results in the lowest structural distance $dS(S', S_I)$ between instance schema $S_I$ and type schema $S'$ (cf. Sect. 2). In this section we show how to automatically derive such *optimal* instance-specific adjustment for unbiased process instances. In the example from Fig. 4, optimal instance-specific adjustment for operation "insert activity $X$ between $A$ and $B$" can be achieved if $X$ is inserted between $A$ and the next possible successor of $A$ which has not been started yet. Theorem 1 captures this aspect:

**Theorem 1 (Optimal Instance-specific Adjustment)** *Let S = (N, E, ...) and S' be two process schemas and let* $\Delta_S$ *= <insert(S, X, A, B)> describe an insertion which transforms S into S'; i.e.,* $S[\Delta_S > S'$. *Let further* $I \in \mathcal{I}$[5] *be an instance running on S for which* $NS_I(B) \in \{$*Running, Completed*$\}$ *holds for the state of activity B; i.e., I is not compliant with S'. We define instance-specific adjustment* $\Delta_I(S')$ *as follows:*

$\Delta_I(S')$ *= move(S',X,A,C) with*
$C \in$ *nextNonStartedSucc(S',I,B)*[6] *where*
*nextNonStartedSucc:* $\mathcal{S} \times \mathcal{I} \times N \mapsto 2^N$ *with*
*nextNotStartedSucc(S',I,n) =*
$\{n \in$ *succ*(S',X)* [7]$\mid NS_I(n) \notin \{$*Running, Completed*$\}$
$\wedge$ ( $\nexists$ *n':* $NS_I(n') \notin \{$*Running, Completed*$\}$ $\wedge\ n \in$ *succ*(S',n')* $\}$
*Then: I is compliant with* $S_I$ *where* $S'[\Delta_I(S')>S_I$ *and*
*dS(S',*$S_I$*) = min*$\{$*dS(S',*$\tilde{S_I}$*)* $\mid S'[\Delta_I\tilde{(}S')>\tilde{S_I}$ *with I compliant with* $\tilde{S_I}\}$

We prove Theorem 1 in a technical report [9]. Its adjustment strategy is applicable for single insert operations with accompanying data flow change operations as well. Consider, for example, Fig. 3 where the insertion of activity $X$ at schema level is accompanied by insertion of a read data edge between activity $X$ and data element $d$. If we adjust the insert operation at instance level to $\Delta_I(S')$ = <move(S',X,A,C)> according to Theorem 1, data flow remainsl correct. This holds for all possible single insert operations with accompanying data flow operations. Accompanying data flow operations for single activity insertions can only be insertions of read data edges for the inserted activity. Correct supply of the corresponding read accesses at schema level requires that the left anchor of the newly inserted activity or one of its predecessors writes the data element accordingly. Since the left anchor as well as all its predecessors still precede the inserted activity after adjusting the change at instance level, the read access will be correctly supplied.

So far, insert and delete operations have been discussed. Regarding insert operations instance-specific adjustments can be applied (cf. Strategy 2), whereas for delete operations, "adjustment" of the execution history of the affected instance might relax compliance issues (cf. Strategy 3). What about move operations? First, moving an activity

[5]$\mathcal{I}$ denotes the set of all instances

[6]In connection with parallel or alternative branchings more than one successor of B can be the "next non-started activity". In this case one of them is chosen arbitrarily or the user is asked to make a choice.

[7]succ*(S, X) returns all direct and indirect successor activities of acitivity X in S.

can be logically seen as combination of a delete and insert operation. Second, regarding state compliance, an activity cannot be moved if it has already been started or completed. Further it cannot be moved in front of an activity which has been already started or completed. Thus we distinguish between the following cases: if an activity shall be moved, which has been already started or completed, we apply history-based adjustment (Strategy 3). If it shall be moved to a position before an already started or completed activity, we apply instance-specific adjustment (Strategy 2).

When using ADEPT technology and AristaFlow BPM Suite in practice, it has often become necessary to apply more than one (i.e., *multiple*) change operation to conduct an intended change at process type level. Assume, for example, that a set of activities is inserted while other ones are deleted at same time. The challenge then is to determine instance-specific adjustments in case of multiple change operations. We illustrate the basic idea of instance-specific adjustments for multiple change operations by example of multiple insertions: For each insert operation applied to schema $S$ a corresponding move operation is generated based on Theorem 1 if necessary; i.e., if I is not compliant regarding this particular insert operation. In Fig. 5, $I$ is not compliant with $S'$. Hence for both insert operations (i.e., of $X$ and $Y$) corresponding move operations are applied to realize an instance-specific adjustment (bias). However, we have to take special care of multiple insertions in connection with data flow changes: assume that for the change operations depicted in Fig. 5 accompanying data flow changes are applied, i.e., new data element $d$ is inserted with write data edge from $Y$ to $d$ and read access from $X$ on $d$. At schema level, these data flow change operations are correctly applicable since $Y$ is a predecessor of $X$ in any case. However, at instance level, after adjusting instance $I$ as shown in Fig. 5, it is not guaranteed that data element $d$ is written by $Y$ before being read by $X$, since $Y$ is no predecessor of $X$ for $I$ on $S'$ (specifically $X$ and $Y$ are ordered in parallel). Thus the correct data supply of $X$ by $Y$ is not ensured at instance level.

Thus, the optimality criterion set out by Theorem 1 has to be modified to *conditional optimality*; i.e., we have to determine instance-specific adjustment $\Delta'_I(S')$ in such a way that the order relation between writing activity $X$ and reading $Y$ is preserved. This can be accomplished by inserting an additional sync edge between $X$ and $Y$ (this is possible since they are ordered in parallel) leading to instance-specific schema $S_{I'}$. We will elaborate on conditional optimality in future work.

## 5 Strategy Evaluation

Assume that instances $\mathcal{I}_S := \{I_1, ..., I_n\}$ are running on type schema $S$ which is then transformed into schema $S'$

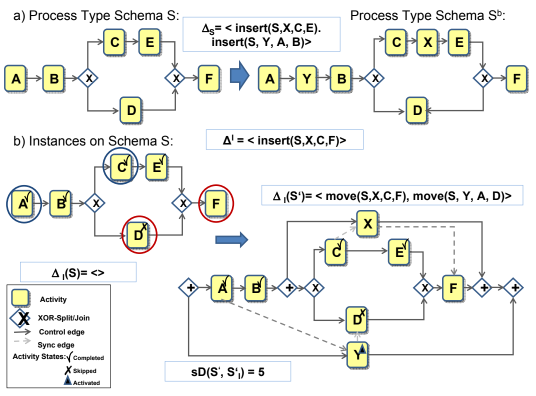

a) Process Type Schema S:
$\Delta_S$= < insert(S,X,C,E). insert(S, Y, A, B)>
Process Type Schema S$^b$:
A
B
C
D
E
F
X
Y
b) Instances on Schema S:
$\Delta^I$ = < insert(S,X,C,F)>
$\Delta_I$(S')= < move(S,X,C,F), move(S, Y, A, D)>
$\Delta_I$(S)= <>
Activity
XOR-Split/Join
Control edge
Sync edge
Activity States: √ Completed
X Skipped
▲ Activated
sD(S', S'$_I$) = 5

**Figure 5. Instance-specific Adjustment**

($S[\Delta_S>$S'). Assume further that a subset of the running instances is not compliant with S'. In this section we want to evaluate the quality of the different strategies introduced in Section 3 for treating such non-compliant instances. On the one hand, the quality of a strategy is related to the number of instances which can be migrated in addition to the number of compliant instances (i.e., instances without applying any strategy). On the other hand, we also have to consider the "price" to be paid for each additionally migrated instance $I$. The latter is reflected by the distance between instance schema $S_I$ and $S'$ after migration. One metric for measuring structural distance between process schemas $S_I := (N_I, CtrlE_I, ...)$ and $S' = (N', CtrlE', ...)$ is provided by structural distance $dS(S', S_I)$ as introduced in Sect. 2. However, this metrics can be only applied if $S_I$ and $S'$ have same activity set; i.e., there exists a bijective mapping between $N_I$ and $N'$ or – in other words – activity coverage between $S_I$ and $S'$ is 100%. Actually, activity coverage can be also used to compare process schemas, for example, in the context of Strategy 1 (cf. Section 3), which leads to $N_I \neq N'$ in most cases. Finally, process schemas $S_I$ and $S'$ can be also compared based on the number of high-level operations necessary to transform $S_I$ into $S'$.

Formula (1) enables evaluation of the quality of Strategies 1 to 4 with $SC(s, S', \mathcal{I}_S) :=$

$$\frac{\#mI_s - \sum_{I\in\mathcal{I}_S} PS(S_I, S') - \sum_{I\in\mathcal{I}_S} AC(S_I, S') - \sum_{I\in\mathcal{I}_S} SD(S_I, S')}{|\mathcal{I}_S|} \tag{1}$$

Specifically, $\#mI_s$ denotes the number of migratable instances when applying strategy $s$. Note that we can compare the application of a strategy with the case of applying no strategy. Then $\#mI_s$ reflects the number of instances, which are compliant with modified type schema version $S'$ anyway. The remainder of the formula quantifies the side-effects when applying one of the strategies. If we do not apply any strategy, the formula yields SC("none", S', $\mathcal{I}_S$)

| Metric M | Formula | Preconditions |
|---|---|---|
| Process Similarity | PS($S_I$,S'):=$\frac{distance(S_I,S')}{\lvert N_I\rvert+\lvert N'\rvert-\lvert N_I \cap N'\rvert}$ [a] | |
| Activity Coverage | AC($S_I$,S'):=$\frac{\lvert N_I \cap N'\rvert}{\lvert N_I\rvert+\lvert N'\rvert-\lvert N_I \cap N'\rvert}$ | |
| Structural Distance | SD($S_I$,S'):=$\frac{dS(S_I,S')}{\lvert N_I\rvert}$ | AC($S_I$,S') = $100\%$ |

[a] distance($S_I$,S')denotes the high-level operations applied for transforming $S_I$ into $S'$.

**Table 2.** *Metrics for Comparing Process schemas*

= $\frac{\#mI_s}{|\mathcal{I}_S|}$; i.e., the number of compliant instances divided by the number of all instances. For Strategy 1, for example, which enables migration of all instances ("always-migrate"), $\#mI_s = |\mathcal{I}_S|$ and thus $SC = 1$ holds.

Consider the scenario depicted in Fig. 6. Type schema $S$ is transformed into $S'$ by inserting activity $X$ between $A$ and $B$ and by deleting $C$. Assume that 500 instances are running based on $S$ and that they are clustered along their current instance state. Assume further that for instances $I_1, ..., I_{100}$, activity $A$ is completed and $B$ is activated, for instances $I_{101}, ..., I_{200}$ activities $A$ and $B$ are completed and $C$ is activated, and for $I_{201}, ..., I_{500}$ activities $A$, $B$, and $C$ are completed whereas $D$ is either activated or running. Then instances $I_1, ..., I_{100}$ are compliant with $S'$, whereas $I_{101}, ..., I_{500}$ have progressed too far; i.e., they are non-compliant with $S'$. Without any further treatment of the non-compliant instances the quality turns out as SC("none",S',$\mathcal{I}_S$) = 0.2.

The result of applying our four strategies is illustrated by Fig. 6. First, Strategy 1 (Always-Migrate) is applied to instances $I_{101}, ..., I_{500}$. Instances $I_{101}, ..., I_{200}$ are not compliant with respect to the insertion of $X$, but they are compliant with respect to the deletion of $C$. Hence, when applying Strategy 1 to $I_{101}, ..., I_{200}$, the insert operation has to be neutralized, which can be expressed by a delete operation on $S'$ (i.e., to delete $X$ on $S'$). Contrary, instances $I_{201}, ..., I_{500}$ are not compliant with respect to both changes. Hence, the deletion of $C$ has to be neutralized as well by a respective insert operation (cf. Fig. 6).

As summarized in Fig. 2, some of the strategies are only applicable in the context of certain change operations; i.e., Strategy 2 for insert operations and Strategy 3 for delete operations. Type schema change $\Delta_S$ captures an insert as well as a delete operation. Hence, we have to combine Strategies 2 and 3 in order to adequately treat non-compliant instances by bias-based local adjustment. Regarding instances $I_{101}, ..., I_{200}$, the delete operation can be applied (no history-based adjustment). However, the insert operation has to be adjusted to a move operation as depicted in Fig. 6. Regarding $I_{201}, ..., I_{500}$ we have to apply history-based adjustment for the deletion of $C$ as well (i.e., logically discarding the entries of $C$ in traces of $I_{201}, ..., I_{500}$). Finally, Fig. 6 shows an example for applying Strategy 4 (Global Adjustment); i.e., by inserting activity $X$ between $C$ and $D$ instead of inserting it between $A$ and $B$, all instances for which $D$ has not been started yet become compliant with $S''$ (e.g., $I_1, ..., I_{450}$). Fig. 6 also depicts quality estimations for Strategies 1, 2, and 4 as depicted in Fig. 6.

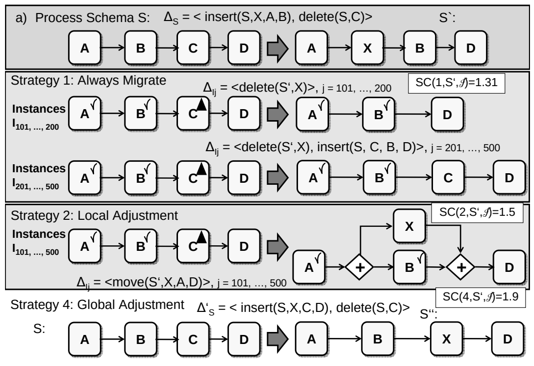

a) Process Schema S: Δ_S = < insert(S,X,A,B), delete(S,C)>
S`:
A B C D A X B D
Strategy 1: Always Migrate
Δ_Ij = <delete(S',X)>, j = 101, ..., 200
SC(1,S',𝒥)=1.31
Instances I_101, ..., 200
Δ_Ij = <delete(S',X), insert(S, C, B, D)>, j = 201, ..., 500
Instances I_201, ..., 500
Strategy 2: Local Adjustment
SC(2,S',𝒥)=1.5
Instances I_101, ..., 500
Δ_Ij = <move(S',X,A,D)>, j = 101, ..., 500
SC(4,S',𝒥)=1.9
Strategy 4: Global Adjustment
Δ'_S = < insert(S,X,C,D), delete(S,C)>
S'':
S:

**Figure 6. Application of Different Strategies**

Aside this quantitative reflections, applicability of the proposed strategies has to be discussed in connection with advanced issues such as loops and semantic constraints on the one side and its application-dependency on the other side. Commenting on the latter, Strategy 1, for example, can be applied when no other strategy seems to be conceivable. Definitely, this might be useful in the medical domain, where there is often no time to roll-back an instance first and then migrate it, but rather let a patient benefit from further process optimizations immediately. Instance-specific and global adjustments exploit the flexibility degrees process schemas offer anyway. This is supported by the emergence of new paradigms such as declarative modeling.

When Strategy 1 is applied within a loop, within the next iteration the change would become effective. However, we have to "take back" the instance-specific bias for subsequent iterations and mark this within the assoicated change operation. We will further investigate on the connection of loops with the other strategies in future work. Regarding semantic constraints, they might yield useful information for the application of the different scenarios. One example is global adjustment which might be applicable without violating soundness, but harming some semantic constraints.

## 6 Related Work

Many approaches deal with soundness issues in adaptive PAIS [12, 14, 8, 11]. The kind of applied correctness criterion often depends on the used process meta model. A discussion and comparison of the particular correctness criteria is given in [8]. Aside from the applied correctness criteria, mostly, these approaches do neither address the question of how to increase the number of migratable instances nor how to deal with non-compliant instances. Frameworks for process flexibility have been presented in [7, 13]. In [7], different paradigms for process flexibility and related technologies are described. [13] provides change patterns and evaluates different approaches based on them. However, [7, 13] do not address the treatment of non-compliant instances. For treating non-compliant instances, different strategies have been proposed in literature (cf. Sect. 2). *Partial rollback* has been suggested [11]. Applying this policy to instances which have already progressed too far results in a compliant state. Generally, instance rollback is accomplished by compensating activities [11]. An obvious drawback is that it is not always possible to find compensating activities, i.e., to adequately rollback non-compliant instances. Apart from this, rollback is mostly connected with loss of work and thus not well accepted by users. An alternative is *delayed migration* of non-compliant instances. Even if instance $I$ on $S$ is not compliant with $S'$ within the actual iteration of a loop, a delayed migration of $I$ to the new change region is possible when another loop iteration takes place. Alternatively, the number of non-compliant instances can be kept smaller by *relaxing* the underlying correctness criteria. Basically, the strategies presented in this paper can be combined with any of the above strategies as well. For example, it is possible to first reduce the number of non-compliant instances by applying relaxed compliance [10], then apply one of the strategies provided in this paper, and try to treat the remaining instances by partial rollback. Further the strategies are applicable to any other flexible process approach using change operations and compliance.

## 7 Summary and Outlook

In this paper we provided four strategies to cope with non-compliant process instances that are based on adjustments either of process changes at type schema level or instance level. Alternatively, "adjustments" of the instance traces are helpful in some cases. All strategies preserve soundness of the running instances after relinking them to the new type schema version. In particular, we elaborated the migration of instances by locally adjusting the type schema change at instance level; e.g., moving the insertion position such that the instances become compliant with the resulting instance-specific schema. How respective adjustments can be determined such that the distance between instance-specific schema and modified type schema version becomes minimal has been shown as well. When considering real-world migration scenarios in domains like healthcare, automotive engineering or electricity supply, we could experience the high practical relevance of the presented research. In future work, we will investigate the interplay between relaxing compliance, treating non-compliant instances, and ensuring semantic process constraints.

## References

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

[2] J. Dehnert and A. Zimmermann. On the suitability of correctness criteria for business process models. In *Int'l Conference Business Process Management*, pages 386–391, 2005.

[3] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *ACM Conf. on Organizational Computing Systems*, pages 10–21, 1995.

[4] C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *Int'l Conf. on Conceptual Modeling*, pages 248–264, 2008.

[5] L. Ly, S. Rinderle-Ma, and P. Dadam. Design and verication of instantiable compliance rule graphs in process-aware information systems. In *Int'l Conf on Advanced Information Systems Engineering*, pages 9–23, 2010.

[6] D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *Int'l Conf. on Advanced Information Systems Engineering (CaISE'08)*, pages 48–63, 2008.

[7] N. Mulyar, M. Schonenberg, R. Mans, N. Russell, and W. van der Aalst. Towards a taxonomy of process flexibility. Technical Report BPM-07-11, BPMcenter.org, 2007.

[8] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.

[9] S. Rinderle-Ma and M. Reichert. Adjustment strategies for non-compliant process instances. Technical Report UIB-2009-06, Ulm University, 2009.

[10] S. Rinderle-Ma, M. Reichert, and B. Weber. Relaxed compliance notions in adaptive process management systems. In *Int'l Conf. on Conceptual Modeling*, pages 232–247, 2008.

[11] S. Sadiq, O. Marjanovic, and M. Orlowska. Managing change and time in dynamic workflow processes. *IJCIS*, 9(1&2):93–116, 2000.

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

[13] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.

[14] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.