Relaxed Compliance Notions in Adaptive Process Management Systems

Stefanie Rinderle-Ma[1], Manfred Reichert[1], and Barbara Weber[2]

[1] Ulm University, Germany
{stefanie.rinderle,manfred.reichert}@uni-ulm.de
[2] University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at

Abstract. The capability to dynamically evolve process models over time and to migrate process instances to a modified model version are fundamental requirements for any process-aware information system. This has been recognized for a long time and different approaches for process schema evolution have emerged. Basically, the challenge is to correctly and efficiently migrate running instances to a modified process model. In addition, no process instance should be needlessly excluded from being migrated. While there has been significant research on correctness notions, existing approaches are still too restrictive regarding the set of migratable instances. This paper discusses fundamental requirements emerging in this context. We revisit the well-established compliance criterion for reasoning about the correct applicability of dynamic process changes, relax this criterion in different respects, and discuss the impact these relaxations have in practice. Furthermore, we investigate how to cope with non-compliant process instances to further increase the number of migratable ones. Respective considerations are fundamental for further maturation of adaptive process management technology.

1 Introduction

The ability to effectively deal with change has been identified as key functionality for any process-aware information systems (PAISs). Through the separation of process logic from application code, PAISs facilitate process changes significantly [1]. In the context of long-running processes (e.g., medical treatment processes [2]), PAISs must additionally allow for the propagation of respective changes to ongoing process instances. Regarding the support of such dynamic process changes, PAIS robustness is fundamental; i.e., dynamic changes must not violate soundness of the running process instances. This cannot be always ensured, for example, when "changing the past" of an instance. As example consider Fig. 1 where change Δ inserts two activities X and Y together with a data dependency between them. Applying Δ to instance I could lead to a situation where Y is invoked though its input data has not been written by X. Another challenge in the context of dynamic process changes concerns the treatment of the dynamic change bug [3]; i.e., the problem to correctly adapt process instance states (e.g., markings in a Petri Net) when performing a dynamic change.

Q. Li et al. (Eds.): ER 2008, LNCS 5231, pp. 232–247, 2008.

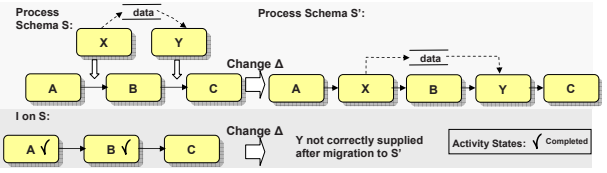


Fig. 1. Changing the Past

In response to these challenges adaptive PAISs have emerged, which allow for dynamic process changes at different levels [4,5,6,7,8,9,10]. Most approaches apply a specific correctness notion to ensure that only those process instances may migrate to a modified process schema for which soundness can be ensured afterwards. One of the most prominent criteria used in this context is *compliance* [4,11]. According to it, a process instance may migrate to schema S' if it is compliant with S'; i.e., the current instance trace can be produced on S' as well. Different techniques have been introduced to efficiently implement this compliance criterion [11,12]. Unfortunately, traditional compliance has turned out to be too restrictive, particularly in connection with loop structures or uncritical changes. Consequently, a large number of instances is excluded from being migrated to a modified schema, even if this does not violate soundness.

In this paper we relax the traditional compliance criterion in different respects, introduce new compliance classes and their properties, and discuss the impact the different relaxations have in practice. Orthogonally, data flow consistency in the context of compliance is discussed. Furthermore, we investigate how to cope with non-compliant process instances to further increase the number of migratable instances. In this context we extent existing approaches [11,8] based on traditional compliance. Altogether respective considerations are fundamental for further maturation of adaptive process management technology.

Section 2 introduces background information needed for the understanding of our work. In Section 3 we revisit the compliance criterion as introduced in [4,11], show how it can be relaxed in different ways to increase the number of migratable instances, and discuss the properties of the resulting compliance classes. Section 4 deals with the handling of non-compliant instances and presents different policies in this context. In Section 5 we extend our considerations to the data flow perspective. An example is given in Section 6. We discuss related work in Section 7 and conclude with a summary and outlook in Section 8.

2 Backgrounds

For each business process to be supported (e.g., handling a customer request or processing an insurance claim) 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 addition, a process schema comprises sets of data elements and data edges. A *data edge* links an activity with a *data element* and represents a read or write access of this activity to the respective data element. Based on process schema S at run-time new *process instances* can be created and executed. Start or completion events of the activities of such instances are recorded in *traces*. WIDE, for example, only records completion events [4], whereas ADEPT distinguishes between start and completion events of activities [11].

Definition 1 (Trace). *Let $\mathcal{PS}$ be the set of all process schemas and let $\mathcal{A}$ be the total set of activities (or more precisely activity labels) based on which process schemas $S \in \mathcal{PS}$ are specified (without loss of generality we assume unique labeling of activities). Let further $\mathcal{Q}_S$ denote the set of all possible traces producible on process schema $S \in \mathcal{PS}$. A particular trace $\sigma_I^S \in \mathcal{Q}_S$ of instance I on S is defined as $\sigma_I^S = < e_1, \ldots, e_k >$ (with $e_i \in$ {Start(a), End(a)}, $a \in \mathcal{A}$, $i = 1, \ldots, k$, $k \in \mathbb{N}$) where the temporal order of e_i in σ_I^S reflects the order in which activities were started and/or completed over S.*[1]

Adaptive process management systems are characterized by their ability to correctly and efficiently deal with *(dynamic) process changes* [12]. Before discussing different levels of change, we give a definition on the topology of change.

Definition 2 (Process Change). *Let $\mathcal{PS}$ be the set of all process schemas and let S, $S' \in \mathcal{PS}$. Let further $\Delta = <op_1, \ldots, op_n>$ denote a process change which applies change operations op_i, i=1,. . . ,n sequentially. Then:*

1. *$S[\Delta> S'$ if and only if Δ is correctly applicable to S and S' is the process schema resulting from the application of Δ to S (i.e., $S' \equiv S + \Delta$)*
2. *$S[\Delta>S'$ if and only if there are process schemas $S_1, S_2, \ldots, S_{n+1} \in \mathcal{PS}$ with $S = S1$, $S' = S_{n+1}$ and for $1 \leq i \leq n$: $S_i[\Delta_i>S_{i+1}$ with $\Delta_i = (op_i)$*

In general, we assume that change Δ is applied to a *sound* (i.e., *correct*) process schema S [13]; i.e., S obeys the correctness constraints set out by the particular process meta model (e.g., bipartite graph structure for Petri Nets). This is also called *structural soundness*. Furthermore, we claim that S' must obey *behavorial soundness* (i.e., any instance on S' must not run into deadlocks or livelocks). This can achieved in two ways: either Δ itself preserves soundness by formal pre-/post-conditions (e.g., in ADEPT [7]) or Δ is applied and soundness of S' is checked afterwards (e.g., by reachability analysis for Petri Nets).

Basically, changes can be triggered and performed at the process type and the process instance level. Changes to a process type T may become necessary to cover the evolution of real-world business processes captured by process schema of this type [9,11,10]. Generally, process engineers can accomplish process type changes by applying a set of change operations to the current schema version S of type T [14]. This results in a new schema version S' of T. Execution of future process instances is usually based on S'. In addition, for long-running instances it is often desired to migrate them to the new schema S' in a controlled and

[1] An entry of a particular activity can occur multiple times due to loopbacks.

efficient manner [11,12]. By contrast, changes of individual process instances are usually performed by end users. They become necessary to react to exceptional situations [7]. In particular, effects of such changes must be kept local, i.e., they must not affect other instances of same type. In both cases, structural and behavioral soundness have to be preserved. The former can be guaranteed since the underlying process schema has to be structurally correct again [11]. The latter, however, has to be explicitly checked. This is accomplished by certain correctness criteria which are subject to the following sections.

3 Revisiting Instance Compliance in Adaptive PAISs

Problems such as dynamic change bug (cf. Sect. 1) show that it is crucial to provide adequate correctness criteria in connection with dynamic process changes. Basically, the challenge is to correctly and efficiently migrate process instances to a modified schema. In particular, no instance should be unnecessarily excluded from such migration except this would lead to severe flaws (i.e., violation of soundness) later on. We first summarize fundamental requirements any correctness notion for dynamic process change should fulfill. Let S be the process schema which is transformed into another schema S' by change Δ; i.e., $S[\Delta > S'$.

Req. 1: Any criterion should guarantee correct execution of process instances on S after migrating them to S'; i.e., soundness has to be preserved; e.g., by ensuring correctly supplied inputs and correct instance states afterwards [12].
Req. 2: The criterion should be generally valid; i.e., it should be applicable independent of a particular process meta model.
Req. 3: The criterion should be implementable in an efficient way.[2]
Req. 4: The number of process instances running on S, which can correctly migrate to S', should be maximized.

Following considerations start with the *compliance criterion* which is a widely used correctness notion [4]. A detailed comparison of compliance and other correctness criteria can be found in [12]. In [12,15] it has been shown that this criterion guarantees Req. 1. Furthermore it presumes no specific process meta model, but is based on traces. Thus Req. 2 is fulfilled as well [11]. In addition, compliance can be checked for arbitrary change patterns [1,14], contrary to criteria which are only valid in connection with a restricted set of change patterns [9]. We have also demonstrated that it can be implemented efficiently [11,12] (cf. Req. 3). However, the traditional compliance criterion does not adequately deal with Req. 4; i.e., it needlessly excludes certain instances from being migrated, though this would be possible without affecting soundness. We relax this criterion by introducing different compliance classes to increase the number of migratable instances. Usually, one cannot decide on such relaxation automatically, but has to consider the particular application context as well. However,

[2] A discussion on the efficiency of correctness checks and a comparison of existing correctness criteria can be found in [12]. In the context of compliance, for example, it should be avoided to access whole trace information for each instance.

the possibility to choose between different compliance classes and to relax correctness constraints on demand enables us to provide advanced user support in connection with process schema evolution.

3.1 Compliance Class TC: Traditional Compliance

The essence of the following criteria is the notion of *compliance*:

Definition 3 (Compliance). *Let S, $S' \in \mathcal{PS}$ be two process schemas. Further let I be a process instance running on S with trace σ_I^S. Then: I is compliant with S' iff σ_I^S can be replayed on S'; i.e., all events logged in σ_I^S could also have been produced by an instance on S' in the same order as set out by σ_I^S.*

In the context of process change, compliance can be used as basis for the following correctness criterion:

Compliance Criterion 1 (Traditional Compliance TC). *Let S be a process schema and I be an instance on S with trace σ_I^S. Let further S be transformed into another schema S' by change Δ; i.e., $S[\Delta>S'$. Then: If I is compliant with S' (cf. Def. 3), this instance can correctly migrate to S'. Specifically, the instance state of I on S' can be logically obtained by replaying σ_I^S on S'. This state is correct again [12,15].*

Compliance Crit. 1 fulfills Req. 1–3 since it forbids changes not compliant with instance histories (reflected by their traces). In special cases, changes of already passed regions do not affect traces and are therefore not prohibited [11,12]. Assume, for example, that at process schema level activity X is inserted into a branch of an alternative branching. If this branch is skipped for a particular instance I at runtime, I will be compliant with the new schema even though its execution has passed the insertion point of X. Reason is that activities of the skipped branch and X do not write any entries into σ_I^S. Therefore trace σ_I^S can be replayed on S'; i.e., I is compliant with the modified schema.

Crit. 1 does not meet Req. 4 in a satisfactory way since it is too restrictive in several respects. Often instances are excluded from migration to the new schema version even though this would not lead to violation of soundness. Consider, for example, changes applied to loops. Even if an instance is compliant within the current loop iteration, according to Crit. 1 it will be considered as non-compliant, if at least one loop iteration took place. Thus, in the following we investigate how traditional compliance can be relaxed to allow for more migratable instances.

3.2 Compliance Class LTC: Loop-Tolerant Compliance

Crit. 1 will unnecessarily restrict the number of migratable instances if the intended process change affects loop constructs as the following example shows:
Example (Restrictiveness of Crit. 1 in conjunction with loops) Consider process schema S from Fig. 2a and assume that activity X is inserted between activities A and B (situated within a loop construct). Assume that instance I has trace σ_I^S as shown in Fig. 2b. Following Crit. 1 change Δ cannot be propagated to I since

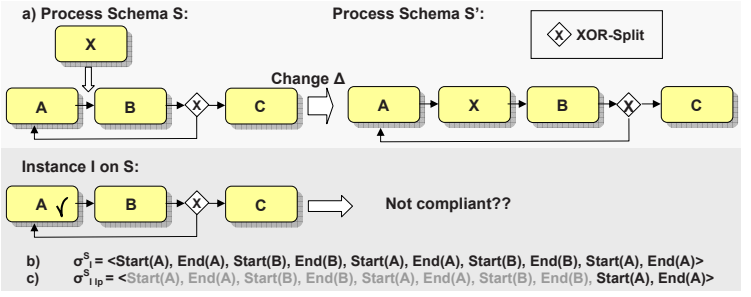


Fig. 2. Process Change Affecting Loop Construct

no trace entries for X have been written in the first two (already completed) iterations of the loop within σ_I^S. According to Crit. 1, therefore, I is considered as being non-compliant with new schema S' even though migration of I to S' would not violate soundness. Consequently, using Crit. 1 only instances which are in the first iteration of the loop construct might be compliant with S'.

In most practical cases it would be too restrictive to prohibit change propagation for in-progress or future loop iterations only because their previous execution is not compliant with the new schema. Think of, for example, medical treatment cycles running for months or years [2]. Any process management system which does not allow to propagate such schema changes (e.g., due to the development of a new medical drug) to already running instances (e.g., related to patients expecting an optimal treatment) would not be accepted by medical staff [2]. Therefore, we have to improve the representation of σ_I^S in order to exterminate its current restrictiveness in conjunction with loops. The key to solution is to differentiate between completed and future executions of loop iterations. From a formal point of view there are two possibilities. The first approach (*linearization*) is to logically treat loop structures as being equivalent to respective linear sequences. Doing so allows us to apply Crit. 1 (with full history information). However, this approach has an essential drawback – explosion of graph size. Thus we adopt another approach which works on a *projection on relevant trace information*, i.e., it maintains the loop construct, but restricts necessary evaluation to relevant parts of the trace. In this context, relevant information includes the actual state of a loop body, but excludes all data about previous loop iterations (cf. Fig. 2c). Note that the projection on relevant information does not physically delete the information about previous loop iterations, but logically hides them (i.e., traceability is not affected).

To realize the desired projection we logically discard all entries from the instance trace produced by a loop iteration other than the actual one (if the loop is still executed) or the last one (if the loop execution has been already finished). For the sake of simplicity we presume nested loops here. However, the described projection can be obtained for arbitrary loop structures as well. We denote this logical view on traces as the *loop-purged trace*.

Definition 4 (Loop-purged Trace). *Let $S \in \mathcal{PS}$ be a process type schema and $\mathcal{A}$ be the set of activities based on which schemas are specified. Let further I be a process instance running on S with trace $\sigma_I^S =< e_0, \ldots, e_k >$ (with $e_i \in$ {Start(a), End(a)}, $a \in \mathcal{A}$, $i = 1, \ldots, k$, $k \in \mathbb{N}$). The loop-purged trace $\sigma_{I\,lp}^S$ can be obtained as follows: In absence of loops $\sigma_{I\,lp}^S$ is identical to σ_I^S. Otherwise, $\sigma_{I\,lp}^S$ is derived from σ_I^S by discarding all entries related to loop iterations other than the last one (completed loop) or the actual one (running loop).*

Based on this, we define the notion of *loop compliance*:

Compliance Criterion 2 (Loop-tolerant Compliance LTC). *Let S be a process schema and I be a process instance on S with trace σ_I^S. Let further S be transformed into another schema S' by change Δ; i.e., $S[\Delta > S'$. Then: We will denote I as loop-tolerant compliant with S' if the loop-purged trace $\sigma_{I\,lp}^S$ of I can be replayed on S'. If I is loop-compliant with S', it can correctly migrate to S'.*

As shown in [15], Crit. 2 fulfills Req. 1 – 3. In addition, it potentially increases the number of migratable instances when compared to Crit. 1. Thus it contributes to Req. 4. In Sect. 3.4 we measure the effects of switching from Compliance Class TC to Compliance Class LTC.

3.3 Compliance Class RLC: Relaxed Loop-Tolerant Compliance

Further relaxation of Compliance Class LTC (cf. Sect. 3.2) can be achieved when exploiting the semantics of the applied change. Specifically, certain changes (e.g., deleting activities) can be applied independently of the particular instance traces since their application does not affect behavorial soundness of instances. Contrary, inserting or moving activities within completed instance regions might affect behavorial soundness (e.g., causing deadlocks or livelocks). Consider Fig. 3a: Schema S is transformed into schema S' by applying change Δ. More precisely, Δ deletes two activities with a data dependency between them (in practice, for example, the first deleted activity could collect some customer data, while the second one just checks this data). Taking Crit. 1, instance I1 is compliant with S' whereas I2 is not; i.e., I2 is excluded from migration to S'. However, migrating I2 to S' would not result in any violation of soundness; i.e., the state of I2 on S' would be correct and no deadlocks or livelocks would occur.

How to reflect the deletion of already completed activities within instance traces? To preserve traceability, entries of such activities cannot be just physically deleted from traces. Instead, we logically discard them from traces (as for the loop-purged trace representation):

Definition 5 (Delete-purged Trace). *Let $S \in \mathcal{PS}$ be a process schema and $\mathcal{A}$ be the set of activities based on which schemas are specified. Let further I be an instance running on S with trace $\sigma_I^S =< e_0, \ldots, e_k >$ (with $e_i \in$ {Start(a), End(a)}, $a \in \mathcal{A}$, $i = 1, \ldots, k$, $k \in \mathbb{N}$). Assume that a sound schema S is changed into another sound process schema S' by change Δ (i.e., $S[\Delta > S'$). The delete-purged trace $\sigma_{I\,dp}^S$ is obtained as follows: If Δ does not contain any delete operations $\sigma_{I\,dp}^S$ is identical to σ_I^S.*

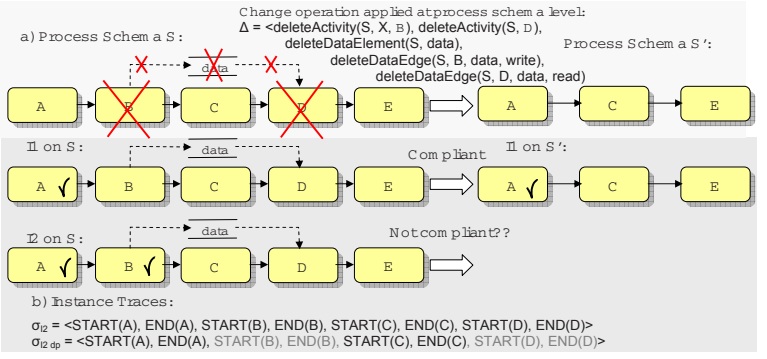


Fig. 3. Changing the Execution History of Process Instances – Example

Otherwise, $\sigma^S_{I\,dp}$ is derived from σ^S_I by (logically) discarding all trace entries related to activities deleted by Δ. Note that $\sigma^S_{I\,dp}$ can be produced on basis of loop-purged trace $\sigma^S_{I\,lp}$ as well (denoted by $\sigma^S_{I\,lp,dp}$).

Based on delete-purged and loop-purged traces, we define the notion of *relaxed (loop-tolerant) compliance*:

Compliance Criterion 3 (Relaxed Loop-tolerant Compliance RLC). *Let S be a schema and I be an instance on S with trace σ^S_I. Let further S be a sound schema which is transformed into another sound schema S' by change Δ; i.e., $S[\Delta > S'$. Then: We denote I as relaxed loop-tolerant compliant with S' if the loop-purged and delete-purged trace $\sigma^S_{I\,lp,dp}$ of I can be replayed on S'. If I is relaxed loop-tolerant compliant, it can correctly migrate to S'.*

Traceability of delete operations can be realized using flags or time stamps as well. Consider the example depicted in Fig. 3b. Start/end events of the deleted activities are not physically deleted from σ_{I2} but logically discarded. Thus, it still can be seen from $\sigma^S_{I_{dp}}$ that activities B and C had been executed before, but then were deleted. This is a different semantics from rolling back activities since effects of the deleted activities are still present (no compensation activities are applied). Based on $\sigma^S_{I_{dp}}$, I2 becomes compliant with S'. Thus the number of compliant instances can be increased again (cf. Req. 4). Though Crit. 3 preserves soundness of affected instances, it depends on the particular application scenario whether it should be applied or not. In any case, based on the above considerations we are able to identify relaxed loop-compliant instances and report them accordingly. Final decision can be left to the process engineer.

3.4 Relation between Compliance Classes

Fig. 4 shows the different compliance classes discussed before. Obviously, the number of compliant instances increases the less restrictive the compliance criterion becomes. At the same time, the number of non-compliant process instances decreases. Formally:

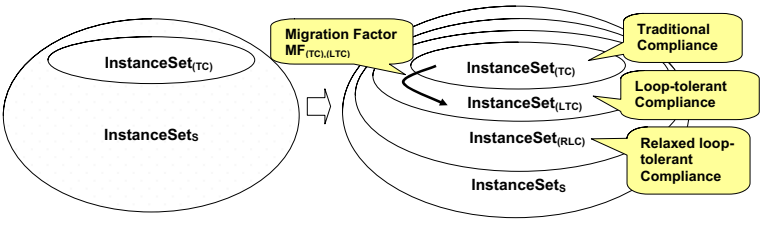


Fig. 4. Compliance Classes

Proposition 1 (Relation between Compliance Classes). *Let S be a sound process schema and $InstanceSet_S$ be a collection of instances running on S . Let further Δ be a change which transforms S into another sound process schema S'. We denote the set of instances which are compliant with S' based on compliance class $CClass \in \{(TC), (LTC), (RLC)\}$ as $InstanceSet_{CClass}$. Then:*

$$InstanceSet_{(TC)} \subseteq InstanceSet_{(LTC)} \subseteq InstanceSet_{(RLC)} \subseteq InstanceSet_S$$

To measure effects when relaxing a compliance class (e.g., TC to LTC), we use the following metrics:

Definition 6 (Migration Factor). *Assumptions as in Prop. 1. Then: The increase in number of instances which can migrate to S' when going from compliance class CClass1 to compliance class CClass2 ((CClass1, CClass2) $\in$ {(TC, LTC), (LTC, RLC), (TC, RLC)}) can be measured by the migration factor*

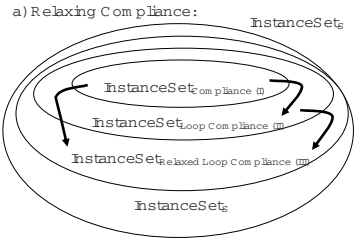
$$MF_{CClass1,CClass2} = \frac{||InstanceSet_{CClass1}| - |InstanceSet_{CClass2}||}{|InstanceSet_S|} \quad (1)$$

4 On Dealing with Non-compliant Process Instances

Even though it is possible to increase the number of compliant instances by switching to the next higher compliance class, the question remains how to deal with *non-compliant* instances. At minimum it is required that non-compliant instances may finish execution according to the schema they were started on or migrated to earlier. In many cases, however, it is desired to allow instances to migrate to the new process schema even though they are not compliant at first sight. For example, this can be crucial in the context of new legal regulations. Generally, it is desired to let as many instances as possible take benefit from future process schema changes. This refers to currently applied optimizations, but also to future ones (applied to the newly designed schema later on).

4.1 Relaxing Compliance

One possibility to deal with non-compliant instances is to relax the underlying compliance criterion. This means to move instances from a stricter compliance class to a relaxed one (cf. Fig. 5a). The effect of doing so can be measured by the migration factor (cf. Def. 6). If relaxation of the compliance class is not possible, non-compliant instances will have to be treated within their current compliance class (cf. Fig. 5b). We discuss different possibilities in the following.



b) Treating within Compliance Class:

Compliance Class/ Method	I	II	III
Partial Rollback	x	x	
Delayed Migration		x	x
Adjusting Changes	x	x	

Fig. 5. Strategies for Treating Non-Compliant Instances

4.2 Treatment within One Compliance Class

We present different strategies for treating non-compliant instances within their particular compliance class; i.e., instances for which their execution "has proceeded too far". As illustrated in Fig. 5b, it depends on the kind of compliance class whether the application of a particular strategy makes sense. Furthermore, the applicability of the following strategies also depends on the semantics of the applied change operation. Altogether, based on the classification presented in Fig. 5b, the adaptive PAIS might suggest the following treatment strategies for non-compliant instances.

Partial Rollback. Several approaches from literature suggest restoring compliance of non-compliant instances by partially rolling them back in their execution [8,16]; i.e., applying this policy for instances which have progressed too far results in a compliant state. Thus a partial rollback is reasonable for compliance classes TC and LTC since both are based on instance states. Contrary, the essence of compliance class RLC is based on allowing changes of the past (specifically delete operations). Hence, rollback to earlier instance states does not make sense here. Generally, (partial) rollback of instances is connected with compensating activities [8] (e.g., if a flight has been booked, the compensating activity will be to cancel the booking). An obvious drawback is that it is not always possible to find compensating activities, i.e., to adequately rollback non-compliant instances. Furthermore, even if compensating activities can be found, this will be mostly connected with loss of work and thus will not be accepted by users.

Delayed Migration. An alternative approach to deal with a non-compliant instance is to wait until it becomes compliant again: Assume that process change Δ affects a loop construct[3] within schema S. Assume further that for instance I running on S this loop is currently being executed, but has proceeded too far to be compliant. However, instance I becomes a candidate for migration when the loop enters its next iteration; i.e., (relaxed) loop-tolerant compliance might be satisfied with delay (*delayed migration*). Such instances can be held as "pending to migration" until the loop condition is evaluated. As we have learned in ADEPT2, implementing delayed migration is not as trivial as it looks like at

[3] Thus delayed migration is applicable for compliance classes LTC and RLC.

first glance. At first, if an instance contained regularly or irregularly nested loops several events (loop backs) might exist to trigger the execution of a previously delayed migration. Furthermore, the interesting question remains how to deal with pending instances when further schema changes take place.

Adjusting Change Operations. The above strategies are based on the idea to reset non-compliant instances into a compliant state. Another approach is to adjust the intended change itself instead of the instance states. We illustrate this taking insert operations as example. However, this strategy can be also applied in the context of other change patterns (e.g., move). The idea is to exploit specific semantics of the insert operation [14]: When applying it, the user has to specify the position where to insert the new activities. Basically, this position depends on two kinds of constraints: first, *data dependencies* have to be fulfilled (e.g., an activity writing data element d has to be positioned before an activity reading d) and second, *semantic constraints* must be obeyed. Here we focus on handling data dependencies. Semantic constraints can be treated similarly.

Basically, adjusting changes can take place at the process type and the process instance level. Assume that a schema S is transformed into another process schema S' by change Δ. Let further I be an instance running on S which is not compliant with S'. If Δ is adjusted to Δ' at type level (transforming S into S''), all instances running on S will be checked for compliance with S'' afterwards (*global adjustment*). Alternatively, Δ can be adjusted specifically for I at instance level. The latter results in *bias* Δ_I; i.e., an instance-specific change which describes the difference between the process schema, I is linked to, and the instance-specific schema it is running on (*instance-specific adjustment*).

Global Adjustment: Consider Fig. 6 where change Δ_1 inserts activities X and Y with a data dependency between them into schema S. This results in schema S'. Instance I running on S is not compliant with S'. Reason is that X would be inserted before already completed activity B. As a consequence, if X is not executed, data will not be written and inputs of Y will not be supplied correctly in the sequel. However, aside any semantic constraints, activity X could be also inserted between activities B and C (Δ_2 transforming S into S''). Reason is that the writing activity (X) is still inserted before the reading one (Y). Thus all data dependencies are still fulfilled. When applying Δ_2, instance I will become compliant with S''.

Generally, more instances will become compliant with a changed process schema, if added activities are inserted "as late as possible". Most important, all data dependencies (or, additionally, semantics constraints) imposed by the process schema and the intended change must be fulfilled. For the given example this implies that activities X and Y can be inserted "later in the process schema" (i.e., as close to the process end as possible) as long as the data dependency between them is still fulfilled. Since a process schema might contain more than one process end node, the formalization of "later in a process schema" should not be based on structural properties; i.e., we aim at being independent of a particular process meta model. As for the compliance criterion, we use process traces in this context. Due to lack of space we omit a formalization here.

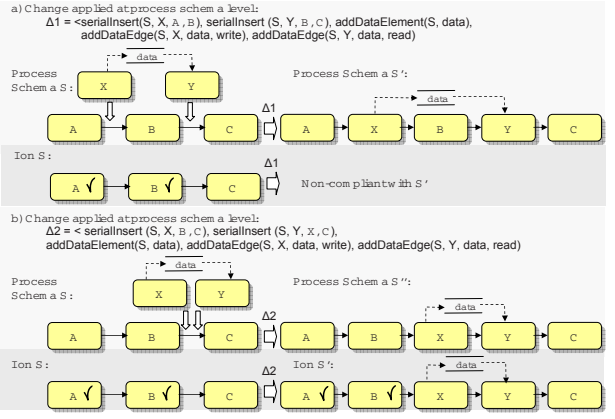


Fig. 6. Global Adjustment of Change Operations – Example

When inserting two or more data-dependent activities as depicted in Fig. 6, 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.

Instance-specific Adjustment: Consider the example depicted in Fig. 7. Contrary to the above example, we do not adjust schema change Δ_1 but apply adjusted instance-specific change $\Delta_I(S)$ only to I at instance level. This results in instance-specific schema S_I. The bias between S_I and S' is captured within $\Delta_I(S')$ and reflects moving X to the position between B and Y.

Instance-specific adjustment can be generalized to make any non-compliant instance compliant with the changed process schema. The idea behind is the following: Let S be a process schema which is transformed into S' by change Δ. Let further I be an instance on S. So far, Δ is propagated to I when migrating I to S (i.e., I reflects Δ after its migration). However, if I is not compliant with S', Δ must not be applied to I. We still can migrate I to S' but without propagating

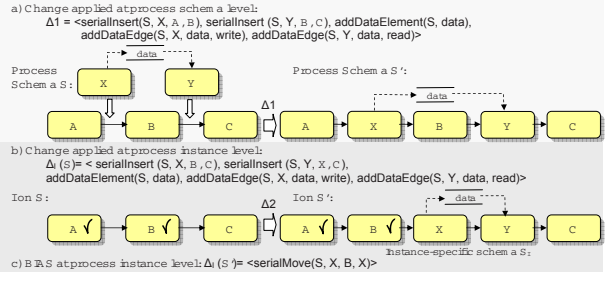


Fig. 7. Instance-Specific Adjustment of Change Operations – Example

Δ to I. This can be achieved by storing an instance-specific bias $\Delta_I(S')$ which has to be calculated; e.g., if Δ inserts activity X at schema level, $\Delta_I(S')$ will contain the "inverse" delete operation of X.

5 The Data Consistency Problem

So far, we have focused on relaxing compliance notions based on the underlying instance traces to increase the number of migratable instances. For three different compliance classes we have shown that soundness is ensured for affected instances. Having a closer look at data flow issues, however, it can be observed that even Crit. 1 is not restrictive enough in some cases.

Example (Inconsistent Read Data Access): We consider the instance depicted in Fig. 8a. Activity C has been started and therefore has already read data value 5 of data element d_1. Assume now that due to a modeling error read data edge $(C, d_1, read)$ is deleted and new read data edge $(C, d_2, read)$ is inserted afterwards. Consequently, C should have read data value 2 of data element d_2 (instead of data value 5). This inconsistent read behavior may lead to errors if, for example, the execution of this instance is aborted and therefore has to be rolled back. Using any representation of trace σ_I^S as introduced so far (i.e., σ_I^S or $\sigma_{I\ lp,dp}^S$), this erroneous case would not be detected. Consequently, this instance would be classified as compliant.

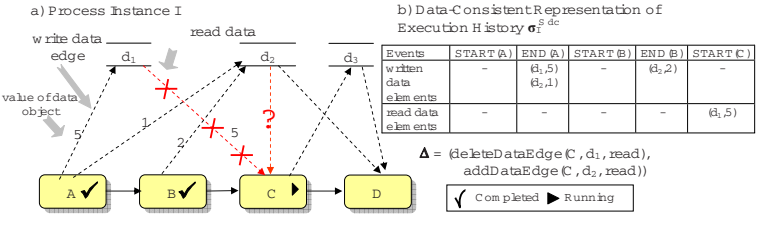


Fig. 8. Data Consistency Problem

We need an adapted form of σ_I^S which also incorporates data flow aspects.

Definition 7 (Data-consistent Trace). *Let the assumptions be as in Def. 1. Let further $\mathcal{D}_S$ be the set of all data elements relevant in the context of schema S. Then we denote σ_I^{Sdc} as data-consistent trace representation of σ_I^S*

with $\sigma_I^{Sdc} = <e_1, \ldots, e_k>$:

$e_i \in \{START(a)^{(d_1,v_1),\ldots,(d_n,v_n)}\ END(a)^{(d_1,v_1),\ldots,(d_m,v_m)}\}$, $a \in \mathcal{A}$

where tuple (d_i, v_i) describes a read/write access of activity a on data element $d_i \in \mathcal{D}_S$ with associated value v_i $(i = 0, \ldots, k)$ if a is started/completed.

Using the data-consistent representation of σ_I^S the problem illutrated in Fig. 8a) is resolved as the following example shows [11,15]:

Example (Consistent Read Data Access Using $\sigma_I^{S^{dc}}$): Consider Fig. 8a. Assume that the data-consistent trace $\sigma_I^{S^{dc}}$ is used instead of σ_I^S. Then the intended data flow change Δ (deleting data edge $(C, d_1, read)$ and inserting data edge $(C, d_2, read)$ afterwards) cannot be correctly propagated to I since entry $\text{Start(C)}^{(d_1,5)}$ of $\sigma_I^{S^{dc}}$ cannot be reproduced on the changed schema.

The data-consistent representation $\sigma_I^{S^{dc}}$ can be used as basis for all other trace representations (cf. Def. 4 – 5). Thus data-consistent compliance works in combination with the other compliance classes TC, LTC, and RLC.

6 Example and Practical Impact

Consider the example depicted in Fig. 9. Schema S is transformed into schema S' by deleting activities B and D and the data dependency between them as well as by inserting activity X within the loop construct. Assume that instances I_k ($k = 1, \ldots, 1000$) are clustered according to their state: For $k = 1, \ldots, 100$, at maximum, activities A, E, and F are completed (indicated by the grey milestone) whereas activities of the other parallel branch have not yet been executed. Particularly, the loop construct is within its first iteration. For $k = 101, \ldots, 200$, the loop has been executed more than once and activities B, C, and D have not yet been executed. For $k = 201, \ldots, 800$, activities of both branches have been executed, but the parallel branching has not completed yet (i.e., G is not activated). For $k = 801, \ldots, 1000$ (not depicted), G is either started or completed.

If Crit. 1 is applied to instances $I_1, \ldots, I_{1000}$, only $I_1, \ldots, I_{100}$ are considered as being compliant with S'. If relaxing to Compliance Crit. 2, additionally instances $I_{101}, \ldots, I_{200}$ become compliant. Thus a migration factor of $MF_{(I),(II)} = 0.1$ is achieved, i.e., 10 % more instances can migrate to S'. Finally, if we relax compliance to Crit. 3, additionally, $I_{201}, \ldots, I_{800}$ are considered as being compliant with S' and a migration factor $MF_{(II),(III)} = 0.6$ results; i.e., 80% of all process instances can migrate to S'. The remaining instances are non-compliant.

7 Related Work

There is a plethora of approaches dealing with correctness issues in adaptive PAISs [9,5,10,17,11,8]. 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 [12]. 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. Most approaches which treat non-compliant instances are based on partial rollback [8,16] (cf. Sect. 4). An alternative approach supporting *delayed migrations* of non-compliant instances is offered by *Flow Nets* [5]. 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.

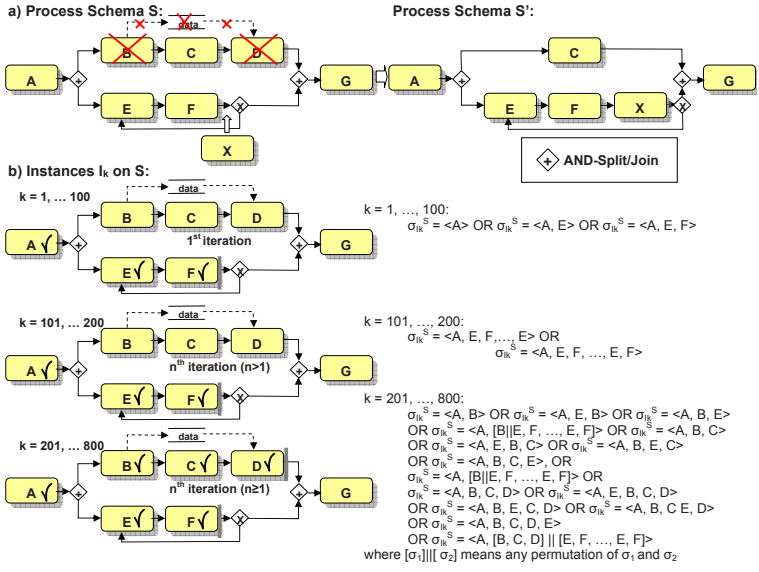


Fig. 9. Example

Frameworks for process flexibility have been presented in [18,14]. In [18], different paradigms for process flexibility and related technologies are described. [14] provides change patterns and evaluates different approaches based on them. However, [18,14] do not address relaxed soundness criteria for process changes.

8 Summary and Outlook

This paper addressed the question of how to increase the number of process instances which can migrate to a changed process schema. This is important in the context of new legal regulations or process optimizations. Thus, we revisited the notion of compliance – a widely-used correctness criterion in the context of process change – and introduced several classes of relaxed compliance. We also showed how the number of compliant instances can be increased by these relaxed notions. Furthermore, we discussed approaches dealing with non-compliant process instances and introduced new strategies in this context. In addition, we detected that traditional compliance is too relaxed in the context of data flow correctness and provided an adequate criterion for data-consistent compliance. Finally we presented a practical example. The concepts of loop-tolerant compliance and data consistency have been implemented in our ADEPT demonstrator [15]. Currently, the concepts are implemented within the full-blown adaptive PAIS ADEPT2. In future work we will investigate the relaxation of compliance more deeply: in addition to further relaxation classes, we will elaborate the strategy of using ad-hoc changes to migrate any non-compliant process instance (without instance-specific changes) to a changed process schema.

References

1. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
2. Lenz, R., Reichert, M.: IT support for healthcare processes – premises, challenges, perspectives. Data and Knowledge Eng. 61(1), 39–58 (2007)
3. van der Aalst, W.: Exterminating the dynamic change bug: A concrete approach to support worfklow change. Information Systems Frontiers 3, 297–317 (2001)
4. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering 24, 211–238 (1998)
5. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COOCS 1995, pp. 10–21 (1995)
6. Kradolfer, M., Geppert, A.: Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In: CoopIS 1999, pp. 104–114 (1999)
7. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. J. of Intelligent Information Systems 10, 93–129 (1998)
8. Sadiq, S., Marjanovic, O., Orlowska, M.: Managing change and time in dynamic workflow processes. IJCIS 9, 93–116 (2000)
9. van der Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoret. Comp. Science 270, 125–203 (2002)
10. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: HICSS-34 (2001)
11. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases 16, 91–116 (2004)
12. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering. 50, 9–34 (2004)
13. Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 386–391. Springer, Heidelberg (2006)
14. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering (2008)
15. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, Ulm University (2004)
16. Reichert, M., Dadam, P., Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Software and Syst. Modeling 2, 37–58 (2003)
17. Rinderle, S., Reichert, M., Dadam, P.: Evaluation of correctness criteria for dynamic workflow changes. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 41–57. Springer, Heidelberg (2003)
18. Mulyar, N., Schonenberg, M., Mans, R., Russell, N., van der Aalst, W.: Towards a taxonomy of process flexibility (extended version). Technical Report BPM-07-11, Brisbane/Eindhoven: BPMcenter.org (2007)