ulm university universität
uulm

Adjustment Strategies for Non-Compliant Process Instances

Stefanie Rinderle-Ma, Manfred Reichert

Ulmer Informatik-Berichte

Nr. 2009-06
März 2009

Adjustment Strategies for Non-Compliant Process Instances

Stefanie Rinderle-Ma and Manfred Reichert

[1]Ulm University, Germany, {stefanie.rinderle, manfred.reichert}@uni-ulm.de

Abstract. Enabling changes at both process type and process instance level is an essential requirement for any adaptive process-aware information system (PAIS). Particularly, it should be possible to migrate a (long-)running process instance to a new type schema version, even if this instance has been individually modified before. Further instance migration must not violate soundness; i.e., structural and behavorial consistency need to be preserved. Compliance has been introduced as basic notion to ensure that instances, whose state has progressed too far, are prohibited from being migrated. However, this also excludes them from further process optimizations, which is not tolerable in many practical settings. This paper introduces a number of 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. We show, for example, how to automatically adjust process type changes at instance level to enable the migration of a higher number of instances. The different strategies are compared and discussed along existing approaches. Altogether, adequate treatment of non-compliant process instances contributes to full process lifecycle support in adaptive PAIS.

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 [1]), PAIS 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. In response to these challenges adaptive PAIS have emerged, which allow for dynamic process changes at different levels [2–6]. 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* [2, 7]. According to it, an instance may migrate to schema S' if it is compliant with S'; i.e., current instance trace can be produced on S' as well. Based on compliance it is ensured that instances, whose state has progressed too far, are prohibited from being migrated.

Consider Fig. 1: Based on process schema S process instances $I_1, ..., I_n$ are running. Then S is transformed into new schema version S' by applying process change Δ_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 $I_1, ..., I_k$ may also benefit from $\Delta_{S'}$ if they are compliant with S''. However, $I_{k+1}, ..., I_n$ are excluded from migration to S'' anyway since they are not running on S'. This is not tolerable in many practical settings. Think, for example, of a patient who is excluded from a new optimized examination because his particular treatment instance cannot be migrated to the changed process type schema. In these cases, it is crucial to find *strategies* to treat non-compliant instances such that they can benefit from further process optimizations as well (cf. Fig. 1).

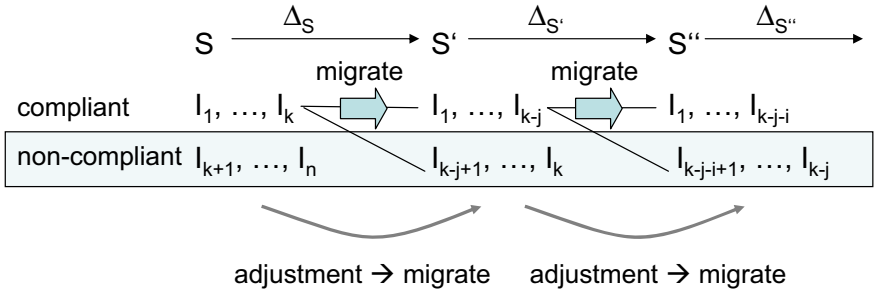

Fig. 1. Migrating Compliant Instances and Treating Non-compliant Ones

In this paper, we introduce a number of 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. We show, for example, how to automatically adjust process type changes at instance level to enable the migration of a higher number of instances. In particular, we do not only consider process instances which are still running according to the process type schema version they were started on (denoted as *unbiased* process instances), but we elaborate adjustment strategies in the context of *biased* process instances as well; i.e., instances which have been already individually modified and thus their instance-specific schema deviates from the original process type schema version. The different strategies are compared and discussed along existing approaches. Altogether the adequate treatment of non-compliant process instances contributes to full lifecycle support

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

of business processes in adaptive PAIS. Section 2 introduces background information. Section 3 presents different strategies for dealing with non-compliant instances. In Section 4, we discuss metrics for measuring the distance between process schemas which is needed for evaluating the different strategies presented in Section 3. Sections 5 and 6 show how non-compliant process instances can be adjusted regardless whether they are unbiased or biased. Section 7 provides an evaluation for applying the different strategies. Section 8 discussed related work and Section 9 closes with a summary and outlook.

2 Backgrounds

2.1 Basic Concepts

In this section we 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 *block-structured* process schemas (like in BPEL). In order to relax strict block structure, we provide additional *sync links* for setting up order relations between activities belonging to parallel branches (similar to the link concept suggested in BPEL). 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. 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 run-time new *process instances* can be created and executed. For an instance I running on S, start and/or completion events of corresponding activities are recorded in *execution trace* σ_I^S. A compact representation of trace σ_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* [7].

Adaptive process management systems are characterized by their ability to correctly and efficiently deal with *(dynamic) process changes* [8]. Before discussing different levels of change, we give a definition on the topology of change.

Definition 1 (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 [9]; i.e., S obeys the correctness constraints set out by the particular process meta model (e.g., block structuring) . This property is also denoted as *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 be achieved in two ways: either Δ itself preserves soundness by formal pre-/post-conditions (e.g., ADEPT [3]) or Δ is applied and soundness of S' is checked afterwards (e.g., by reachability analysis).

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 [5, 7, 6]. Generally, process engineers can accomplish process type changes by applying a set of change operations to the current schema version S of type T [10]. This results in 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 efficient manner [7, 8]. By contrast, *changes of individual process instances* are usually performed by end users and become necessary to react to exceptional situations [3]. 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 [3]. Behavorial soundness, in turn, refers to correct instance states [7]. If, for example, an activity is started before all its predecessor activities are completed, this activity will not necessarily be supplied with all required input data. Behavorial soundness is accomplished by behavorial correctness criteria. In the context of change one prominent example is the *compliance criterion* [2] which is used in the remainder of this paper (a detailed comparison of compliance and other correctness criteria can be found in [8]). Informally, compliance checks whether execution trace σ_I^S of instance I on schema S could also be produced by an instance on S' in the same order as set out by σ_I^S.

2.2 Overall Change Framework

Assume that at process type level schema S evolves to schema S' and that we want to migrate instances on S to S' if they are compliant. Out focus is on how to deal with non-compliant instances in this context. Specifically, we

Change operation Δ 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	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 betweena ctivity sets A and B				

[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 (details see [10]).

[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*

want to find *strategies* to cope with non-compliant instances (i.e., to migrate them to S' without violating soundness) regardless whether they are unbiased or biased. Due to their instance-specific bias, the treatment of non-compliant biased instances poses additional challenges when compared to the treatment of unbiased ones. In order to tackle these challenges in a systematic way, we base following considerations on the classification for biased instances as proposed in [11, 12]. This classification uses the *degree of overlap* between type change and instance-specific bias. Overlapping changes occur if at instance level some or all of the subsequent type changes are anticipated. If Δ_I completely anticipates Δ_S (i.e., $S_I = S'$), we denote the changes as *equivalent*. In this case I can be always migrated to S' without any further check. Changes can also subsume each other. Δ_I subsumes Δ_S (denoted by $\Delta_I \succ \Delta_S$) if all change operations captured by Δ_S are contained within Δ_I, but Δ_I also captures additional change operations. In this case, I is compliant with S' as well. However, new bias $\Delta'_I := \Delta_I \backslash \Delta_S$ for I on S' has to be calculated. In turn, if Δ_S subsumes Δ_I (denoted by $\Delta_S \succ \Delta_I$) it has to be checked whether the changes of Δ_S not present for I so far (i.e., $\Delta_S \setminus \Delta_I$) can correctly applied to I. Finally, Δ_I and Δ_S *partially overlap* if they have some change operations "in common", but each of them also captures additional changes. In this case, it has to be checked whether the changes in $\Delta_S \setminus \Delta_I$ can be correctly applied to I, and new bias $\Delta_I \setminus \Delta_S$ has to be calculated for I on S'.

3 Strategies for Coping with Non-compliant Instances

In this section, we discuss novel strategies for coping with non-compliant instances. The basic idea behind is to conduct certain kinds of adjustment (cf. Fig. 3) to also enable non-compliant instances to migrate to the new schema version; i.e., to be relinked 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 (non-compliant) 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. In the following we present two

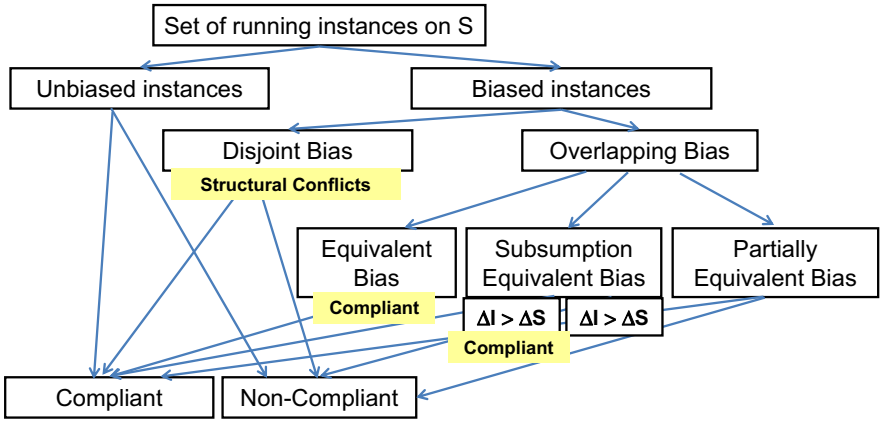

Fig. 2. Overview on Instance Cases along Change Overlap

strategies for this *bias-based* adjustment (cf. Fig. 3). Other adjustments adapt of instance execution traces or even the type change itself.

3.1 Biased-based Local Adjustment

Strategy 1 (Always-Migrate): Basically, **any** non-compliant instance can be migrated (i.e., relinked) to the modified type schema version. The idea is as follows: Let S be a process schema which is transformed into S' by change Δ_S. Let further I be an instance on S. So far, Δ_S is propagated to I when migrating I to S (i.e., I reflects Δ_S after its migration to S'). However, if I is not compliant with S', Δ_S must not be applied to I; i.e., execution of I should be continued on its old schema. However, this effect can be also "simulated" when

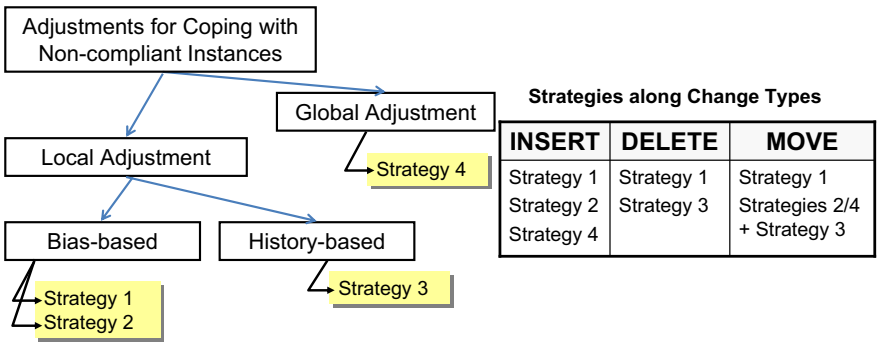

Fig. 3. Overview on Possible Strategies for Coping with Non-Compliant Instances

re-linking I to S' by "neutralizing" type change Δ_S. "Neutralizing" means to introduce an artificial bias Δ_I based on S' which reverses the effects of Δ_S for this particular instance. Consider Fig. 4. At type level new activity X is inserted between activities A and B. Then instance I (cf. Fig. 4b) is not compliant with S' since B is already completed. However, I can still be migrated to to S' by not applying type change Δ_S. The invalid change Δ_S for I on S' can be "healed" by creating an instance-specific bias $\Delta_I(S')$ for I on S'; i.e., $\Delta_I(S')$ reflects the deviations between S' and instance-specific schema S_I. Specifically, $\Delta_I(S')$ constitutes the "inverse" change operation of Δ_S. In our example, insertion of X into S can be reversed by deleting X from S'.

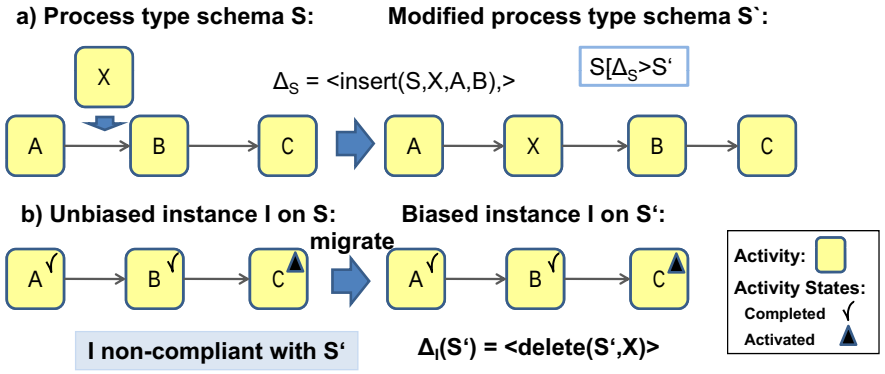

Fig. 4. Strategy 1: Always-Migrate (Example)

Strategy 1 can be applied for any kind of change operation. The challenge is to determine the "inverse" change to be maintained for the migrated instance. Tab. 1 shows the inverse changes for insert, delete, and move operations. Alternatively, we can directly determine schema "difference" between type schema S' and instance-specific schema. Here, already existing approaches can be used. [13], for example, presentw an approach to compare two schemas S_1 and S_2 in terms of high-level change operations (cf. Tab. 1) needed to transform S_1 into S_2.

Strategy 2 (Instance-Specific Adjustment) The idea behind Strategy 2 is to conduct instance-specific adjustments of type changes in order to be able to relink non-compliant instances to the new type schema version. For this purpose we exploit specific semantics of the applied change operation [10]: When applying an insert operation, for example, the user has to specify the position where to insert the new activities (cf. Tab. 1). Thus the insert operation is a candidate for instance-specific adjustments since such position parameters can be easily adapted; e.g., by inserting an activtiy "later" than originally intended. In the following, instance-specific adjustment is elaborated in the context of insert operations. Section 5 also provides a discussion for adjusting move operations.

Consider the example depicted in Fig. 5. Instance I (cf. Fig. 5b) is not compliant with new type schema S' since B is already in state `Running`. Basically, if no semantic constraints are violated, at instance level X can be inserted at other positions as well, specifically at those positions within the instance-specific schema, where I becomes compliant again. Assuming that we keep A as the "left anchor" for the insert operation, possible "right anchors" for the insertion are C and D respectively. Consider first insertion of X between A and C (①): When applying change $\Delta_I^1(S)$ to I as instance-specific adjustment, we obtain instance schema S_I^1. The bias between S_I^1 and S' is then 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. 5b (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 by keeping the instance-specific schemas "as close as possible" to the type schema version they are migrated to. Here, intuitively, S_I^1 is "closer" to S' when compared to S_I^2. In order to formally define "as close as possible" a distance metric between process schemas is needed (cf. Section 4).

Basically, it is also possible to use "left anchors" other than A for realizing instance-specific adjustments. Consider Fig. 5b(③): When compared to S' the order between X and B has been reversed for Fig. 5b(③), whereas for Fig. 5b(①) X and B are ordered in parallel. Thus for Fig. 5b(①), still X can be started before B is finished, what is not possible in Fig. 5b(③). Hence, the instance schema from Fig. 5b(①) is "closest" to S'. – Altogether, from the above discussion two basic research questions can be derived:

1. How to measure the distance between process type and instance schemas?
2. How to determine instance-specific adjustments $\Delta_I(S')$ such that I
   - is compliant with S_I where $S'[\Delta_I(S') > S_I$
   - and the distance between instance schema S_I and modified type schema S' becomes minimal?

For the first question different approaches exist. We discuss them in Section 4. The second question is targeted in Sections 5 and 6.

3.2 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 already running or completed). Basically, deletion of activities "in the past" of process instances can be enabled by adjusting instance traces. Assume, for example, 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 end events for X were logged in σ_I^S. Thus σ_I^S cannot be "replayed" on S' and compliance for I on S'

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

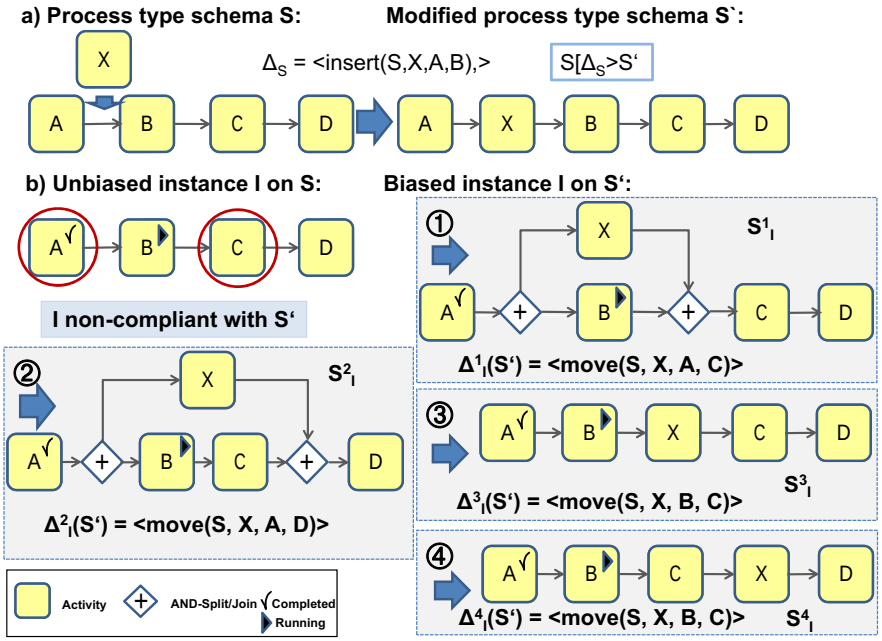

Fig. 5. Strategy 1: Instance-Specific Adjustment (Example)

is not fulfilled. However, when discarding the respective trace entries of X from σ_I^S, the modified trace can be replayed on S' and thus I can be migrated. Entries of deleted activities are not physically deleted from the exeuction traces, but are only flagged within the trace in order to preserve traceability. As shown in [14] Strategy 3 does not harm soundness of the affected instances.

3.3 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 Δ_S. Let further I be an instance on S which is not compliant with S'. If Δ_S is adjusted to Δ'_S at type level (i.e., transforming S into S'' instead of S'), more instances running on S may becomes 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 (or additionally, semantics constraints) imposed by the process type 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 type 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 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 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. 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 Metrics on Process Schemas

As discussed in the context of Strategy 2, measuring the distance between type and instance schema is an important task. Basically, there exist different approaches for this, ranging from activity coverage [15] over fitness functions (e.g., in connection with genetic process mining [16]), to process similarity as defined in [15]. Activity coverage quantifies the distance between two process schemas S_1 and S_2 in terms of the difference between their activity sets.

Regarding the examples depicted in Fig. 5, activity coverage is 100% for all schemas. However, there are structural differences between instance schemas S_I^j (j= 1, .., 4) and S'. First of all, for each S_I^j a bias Δ_I^j between S_I^j and S' arises. According to the process similarity notion introduced in [15], for example, the distance between two process schemas can be determined as number of high-level change operations (i.e., insert, move, and delete operations) needed to transform the one schema into the other. Regarding our scenario from Fig. 5, for each S_I^j exactly one high-level change, specifically a move operation, is required to capture the difference between S' and S_I^j. Using this measure, therefore, the similarity to S' would be the same for all S_I^j (j = 1, ..., 4). Thus, we need an additional measurement in terms of control relations between the different activities. Intuitively, such *structural fitness* can be expressed considering the number of control relations different for S' and S_I^j. This number should be minimized in order to enable propagation of subsequent schema changes.

Definition 2 (Structural Distance). *Let S = (N, CtrlE, SyncE, ...) and S' = (N, CtrlE', SyncE', ...) (cf. Section 2) be two (block-structured) process schemas having equal activity set (i.e., activity coverage of* 100*%, cf. Section 7). Let further succ*(S, n) denote the set of all direct or indirect successors of activity n in S*[3]*, formally:*

succ:* $\mathcal{S} \times N \mapsto 2^N$ *with*

succ(S, n)* $= \{n^* \in N | n^* \in succ(S,n) \vee \exists n^{**} \in succ(S,n) : n^* \in succ^*(S,n^{**}))\}$

with succ(S, n) denoting the set of all direct successors of n in S, i.e., regarding control and sync edges

[3] At this point we assume acyclic process structures. However, in general cyclic structures can contribute to a treatment of non-compliant instances. A sketch of the so called *delayed migration* using loop backward jumps is provided in Section 8.

The structural fitness between S and S' can then be quantified by function sD(S, S') which sums up the cardinalities of the difference sets between the successor sets for all activities contained in S and S'.

sD: $\mathcal{S} \times \mathcal{S} \mapsto \mathbb{N}$ *with*

sD(S,S') $= |\bigcup_{n \in N}(succ^*(S,n) \setminus succ^*(S',n)) \cup (succ^*(S',n) \setminus succ^*(S,n)|$

Fig. 6 summarizes and compares the respective successor sets for the different schemas from Fig. 5. As can be seen instance-specific schema S_I^1 has minimal structural distance to S'. Hence, deciding for an instance-specific adjustment in order to "make" I compliant with S' (cf. Fig. 5b), change Δ_I^1 would be considered as optimal instance-specific adjustment.

A	X B C D	X B C D	X B C D	X B C D	X B C D
X	B C D	C D →	D →	C D →	D →
B	C D	C D	C D	X C D →	X C D →
C	D	D	D	D	X D →
D	∅	∅	∅	∅	∅
D					

Fig. 6. Successor Sets and Structural Distances for the Scenario from Fig. 5

5 Instance-Specific Adjustment of Unbiased Instances

Bias-based strategies as introduced in Section 3.1 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 Δ_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. 5, different adjustments of Δ_S are conceivable. However, we are particularly interested in the adjustment which results in the lowest structural distance $sD(S', S_I)$ between instance schema S_I and type schema S' (cf. Def. 2). In this section we show how to automatically derive such *optimal* instance-specific adjustment for unbiased process instances. In the example from Fig. 5, optimal instance-specific adjustment for operation "insert activity X between A and B" is 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, S' be two process schemas and let* Δ_S = <*insert(S, X, A, B)*> *be an insert operation*

which transforms S into S'; i.e., $S[\Delta_S > S'$. Let further $I \in \mathcal{I}$ ($\mathcal{I}$ denotes the set of all instances) be an instance running on S for which $NS_I(B) \in \{Running, Completed\}$ holds for the state of activity B (cf. Section 2); 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)*[4] *where*
nextNonStartedSucc: $\mathcal{S} \times \mathcal{I} \times N \mapsto 2^N$
nextNotStartedSucc(S',I,n) =
$\{n \in succ^*(S',X) \mid NS_I(n) \notin \{Running, Completed\}$
$\wedge$ ($\nexists$ 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*
$dS(S',S_I) = min\{dS(S',\tilde{S}_I) \mid S'[\Delta_I(S') > \tilde{S}_I$ *with I compliant with* $\tilde{S}_I\}$

A proof of Theorem 1 can be cound in Appendix A.

For each insert operation applied to process 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. As example consider Fig. 7: instance I is not compliant with S'. Hence for both insert operations (i.e., of X and Y) two corresponding move operations are applied to realize an instance-specific adjustment (bias).

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 of all, moving an activity can be seen as combination of a delete and insert operation. Second, regarding compliance, an activity cannot be moved if it has been already started or completed. Furthermore it cannot be moved before 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 the activity is to be moved before an already started or completed activity, we apply instance-specific adjustment (Strategy 2).

6 Instance-Specific Adjustment of Biased Instances

Fig. 2 has already given an overview on the different classes of overlap between schema changes at the process type and the process instance level. In this section we investigate whether and – if yes – how the strategies for dealing with non-compliant instances (cf. Section 3) can be applied in the context of biased instances as well. First of all, one can observe that instances with equivalent bias and instances having subsumption equivalent bias (i.e., $\Delta_I \succ \Delta_S$) are always compliant with the changed schema (cf. Section 2). Consequently, for these two

[4] 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.

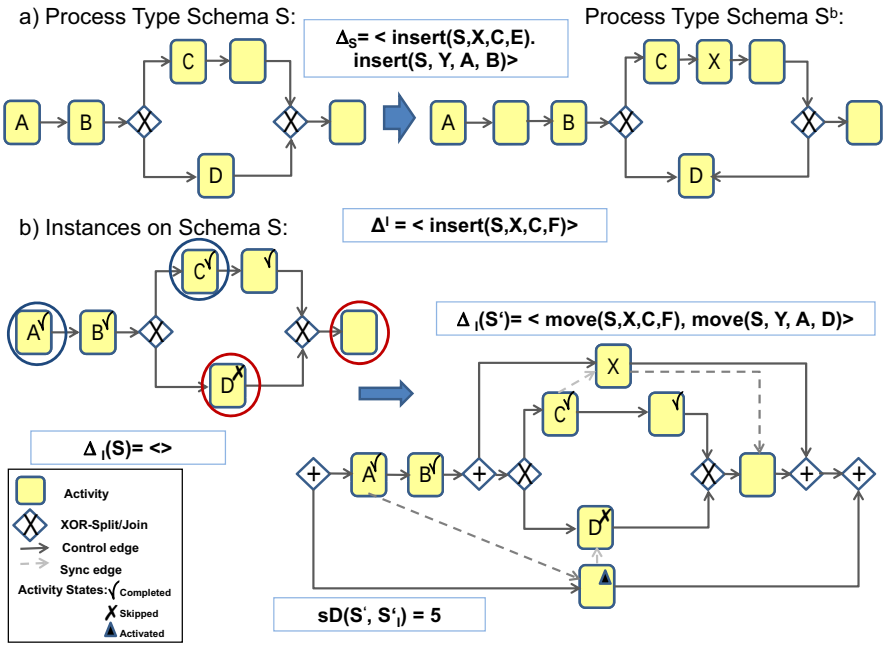

Fig. 7. Instance-specific Adjustment (Example)

classes no additional reasoning on treatment of non-compliant instances becomes necessary. From the remaining overlap classes, we first focus on instances with disjoint bias; i.e., changes at type level and instance level which are completely different. An example is depicted in Fig. 8. Regarding instance I, activity Y has been inserted between B and C (captured by instance-specific bias $\Delta_I(S)$). Then at type level, schema change Δ_S inserts X between A and B. Obviously, Δ_S and Δ_I are disjoint and I is not compliant with S' (since B is already completed). Transferring the instance-specific adjustment as presented in Theorem 1 to this scenario, first of all next successor of B in S_I which has not been started yet is determined; i.e., activity Y. Thus, the instance-specific adjustment of Δ_S at instance level yields move operation move(S', X, A, Y). Since Δ_S and $\Delta_I(S)$ are disjoint, the instance-specific adjustment does not conflict with the original bias. Thus, new bias $\Delta_I(S')$ turns out as concatenation of the bias before migration and the instance-specific adjustment; i.e., $\Delta_I(S')$ as shown in Fig. 8. Due to lack of space we omit formal details and proofs here.

If type schema change Δ_S subsumes instance change Δ_I (i.e., $\Delta_s ucc \Delta_I$) instance-specific adjustment has to be carried out for all insert operations in $\Delta_S \setminus \Delta_I$ for which I has progressed too far. How to determine difference sets between changes is outside the scope of this paper. For all adjusted insert operations the corresponding move operation can be determined based on Theorem 1. These

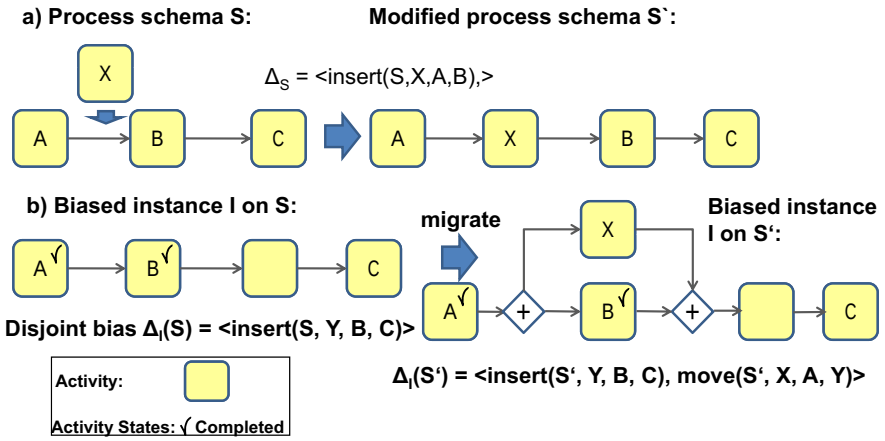

Fig. 8. Instance-specific Adjustment for Instances with Disjoint Bias

move operations have to be stored as instance-specific bias on S' after migration. Fig. 9 shows an example.

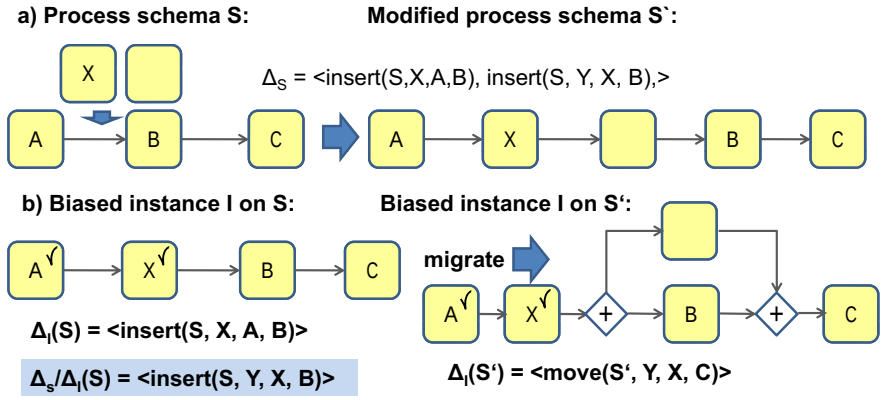

Fig. 9. Instance-specific Adjustment of Instances with Subsumption Equivalent Bias

7 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' ($S[\Delta_S>$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 non-compliant instances. On the one hand, the quality of a strategy is based on the number of instances which can be migrated additionally 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 the structural distance between process schemas $S_I := (N_I, CtrlE_I, ...)$ and $S' = (N', CtrlE', ...)$ is provided in Def. 2. However, this metrics can be only applied if S_I and S' have the same activity sets; 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' (denoted as process similarity [15]). For a definition of all three metrics see Tab. 2:

Metric M	**Formula**	**Preconditions**
Process Similarity	PS(S_I,S'):=$\frac{\#ops}{\lvert N_I\rvert+\lvert N'\rvert-\lvert N_I\cap N'\rvert}$ [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'):=$\frac{sD(S_I,S')}{\lvert N_I\rvert}$	AC(S_I,S') = 100%

[a] ops denotes the high-level operations applied for transforming S_I into S'.

Table 2. *Different Metrics for Comparing Process schemas*

Below Formula (1) enables evaluation of the quality of Strategies 1 to 4. 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', $\mathcal{I}_S$) = $\frac{\#mI_s}{|\mathcal{I}_S|}$; 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"), $\#mI_s = |\mathcal{I}_S|$ holds.

$$SC(s, S', \mathcal{I}_S) := \frac{\#mI_s - \sum_{I \in \mathcal{I}_S} PS(S_I, S') - \sum_{I \in \mathcal{I}_S} AC(S_I, S') - \sum_{I \in \mathcal{I}_S} sD(S_I, S')}{|\mathcal{I}_S|} \quad (1)$$

Consider the scenario depicted in Fig. 10. 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 $I_{101}, ..., I_{200}$ 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. 10. 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 also the deletion of C has to be neutralized by a respective insert operation (cf. Fig. 10).

As summarized in Fig. 3, some of the strategies are applicable in the context of certain change operations; i.e., Strategy 2 for insert operations and Strategy 3 for delete operations. Type schema change Δ_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. 10. Regarding instances $I_{201}, ..., I_{500}$ 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. 10 shows an example for the application of Strategy 4 (Global Adjustment); i.e., by inserting activity X between C and D instead of between A and B all instances, for which D has not been started yet become compliant with S'' (e.g., $I_1, ..., I_{450}$). Altogether, we obtain the following quality estimations for Strategies 1, 2, and 4 as depicted in Fig. 10:

- Strategy 1: S(s1, S', $\{I_1, ..., I_{500}\}$) = $\frac{500-(100*0.25+300*0.4)+(100*0.75+300*0.6)}{500} = 1.31$
- Strategy 2: S(s2, S', $\{I_1, ..., I_{500}\}$) = $\frac{500-500*0.25+500-500*0.25}{500} = 1.5$
- Strategy 4: S(s3, S', $\{I_1, ..., I_{500}\}$) = $\frac{450+500}{500} = 1.9$

8 Related Work

There is a plethora of approaches dealing with correctness issues in adaptive PAIS [5, 17, 6, 7, 4]. 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. Most approaches which treat non-compliant instances are based on partial rollback [4, 18] (cf. Sect. 4). An alternative approach supporting *delayed migrations* of non-compliant instances is offered by *Flow Nets* [17]. Even if instance I on S is

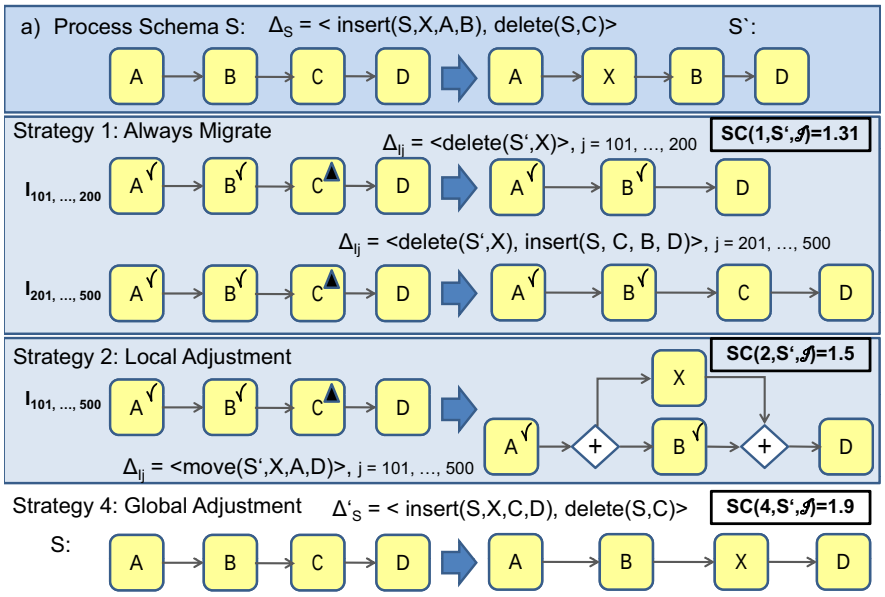

Fig. 10. Application of Different Strategies

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.

Frameworks for process flexibility have been presented in [19, 10]. In [19], different paradigms for process flexibility and related technologies are described. [10] provides change patterns and evaluates different approaches based on them. However, [19, 10] do not address the treatment of non-compliant instances.

For treating non-compliant instances, *partial rollback* has been suggested [4, 18]. Applying this policy for instances which have already progressed too far results in a compliant state. Generally, instance rollback is accomplished by compensating activities [4]. 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.

9 Summary and Outlook

So far, non-compliant process instances have been excluced from being migrated to the new process type schema version in order to preserve soundness. Consequently, these instances are also excluded from any future process optimizations. Thus, in this paper we provided four strategies to cope with non-compliant process instances. Specifically, the strategies 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. Finally, all strategies were evaluated along an example. In future work, we aim at implementing the different strategies together with our framework on relaxing compliance notions within the ADEPT2 system. Furthermore, we plan to investigate the interplay between relaxing compliance, treating non-compliant instances, and ensuring semantic process constraints.

References

1. Lenz, R., Reichert, M.: IT support for healthcare processes – premises, challenges, perspectives. Data and Knowledge Eng. **61** (2007) 39–58
2. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering **24** (1998) 211–238
3. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. J of Intelligent Information Systems **10** (1998) 93–129
4. Sadiq, S., Marjanovic, O., Orlowska, M.: Managing change and time in dynamic workflow processes. IJCIS **9** (2000) 93–116
5. van der Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoret. Comp. Science **270** (2002) 125–203
6. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
7. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116
8. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Eng. **50** (2004) 9–34
9. Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: Int'l Conference Business Process Management. (2005) 386–391
10. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering **66** (2008) 438–466
11. Rinderle, S., Reichert, M., Dadam, P.: On dealing with structural conflicts between process type and instance changes. In: Int'l Conf. on Business Process Management. (2004) 274–289
12. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, Ulm University (2004)
13. Li, C., Reichert, M., Wombacher, A.: On measuring process model similarity based on high-level change operations. In: 27th Int'l Conf. on Conceptual Modeling (ER'08). (2008) 248–264
14. Rinderle-Ma, S., Reichert, M., Weber, B.: Relaxed compliance notions in adaptive process management systems. In: Int'l Conf. on Conceptual Modeling (ER'08). (2008) 232–247

15. Li, C., Reichert, M., Wombacher, A.: Discovering reference process models by mining process variants. In: Int'l Conf. on Web Services. (2008)
16. Alves de Medeiros, A., Weijters, A., van der Aalst, W.: Genetic process mining: An experimental evaluation. Data Mining and Knowl. Discovery **14** (2007) 245–304
17. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: ACM Conf. on Organizational Computing Systems. (1995) 10–21
18. Reichert, M., Dadam, P., Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Software and Syst. Modeling **2** (2003) 37–58
19. 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)
20. Rinderle, S., Reichert, M., Jurisch, M., Kreher, U.: On representing, purging, and utilizing change logs in process management systems. In: Proc. of 4the Int'l Conf. on Business Process Management, September, Vienna, Austria. (2006) 241–256

A Proofs

Theorem 1 (Optimal Instance-specific Adjustment). *Let S, S' be two process schemas and let* Δ_S = <*insert(S, X, A, B)*> *be an insert operation which transforms S into S'; i.e.,* $S[\Delta_S > S'$. *Let further* $I \in \mathcal{I}$ *(*$\mathcal{I}$ *denotes the set of all instances) be an instance running on S for which* $NS_I(B) \in \{$*Running, Completed*$\}$ *holds for the state of activity B (cf. Section 2); 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)*[5] *where*
nextNonStartedSucc: $\mathcal{S} \times \mathcal{I} \times N \mapsto 2^N$
nextNotStartedSucc(S',I,n) =
$\{n \in succ^*(S',X) \mid NS_I(n) \notin \{$*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*
$dS(S',S_I) = min\{dS(S',\tilde{S_I}) \mid S'[\Delta_I(S')>\tilde{S_I}$ *with I compliant with* $\tilde{S_I}\}$

For proofing Theorem 1, we use the following two Lemmata: Lemma 1 provides an estimation for the structural difference between a modifed type schema version and an instance-specific schema, if the assumption of Theorem **??** hold. Again under the assumptions of Theorem 1, Lemma 2 gives an estimation for the structural difference when the instance-specific change is modified by choosing another left anchor for the move operation at instance level as suggested for the instance-specific change with minimal structural difference.

Lemma 1 (Structural Distance for Instance-Specific Adjustment (1)). *Let the assumptions be as defined for Theorem 1. Then:*
$sD(S_I,S') = |succ^*(S', A) \cap pred^*(S', C)|$*-1*

[5] 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.

Proof of Lemma 1: For illustration see Fig. 11.

The only activity within S_I which is affected by Δ_I in terms of changes of its successor set is X when compared to S'. Reason is that for "left anchor" A (and all its predecessors) the set of successors remains the same (i.e., X is only inserted at a different position). Symmetrically, for "right anchor" C (and all its successors) their successor sets are not affected by $\Delta_I(S')$ at all. Thus, the possibly affected activities are within set $succ^*(S_I, A) \cap pred^*(S_I, C)$.

When digging deeper we obtain the following results: For activities in set $(succ^*(S_I, A) \cap pred^*(S_I, X))$ their successor sets remain equal as well when compared to S'. Note that X is actually present in their successor sets and the exact position does not matter. For all activities in $succ^*(S_I, X) \cap pred^*(S_I, C)$, X is inserted in parallel. Assume that A and B are direct successors in S and X is inserted serially between A and B resulting in S'. Since B has been already started for I, X is moved between A and a successor of B. Thus, this always results in inserting X at least parallel to B. Altogether, for $succ^*(S_I, X) \cap pred^*(S_I, C)$ there is also no change in their successor sets when compared to S' since no new order relations are introduced for these activities.

In summary, only the successor set of X is affected when comparing S_I and S'. As argued before, X is moved parallel to the activities which are in set $succ^*(S', A) \cap pred*(S', C)$ except X itself. Thus, for all these activites (except X) X "looses" one successor when comparing S' and S_I. Out of this statement Lemma 1 can be directly concluded. □

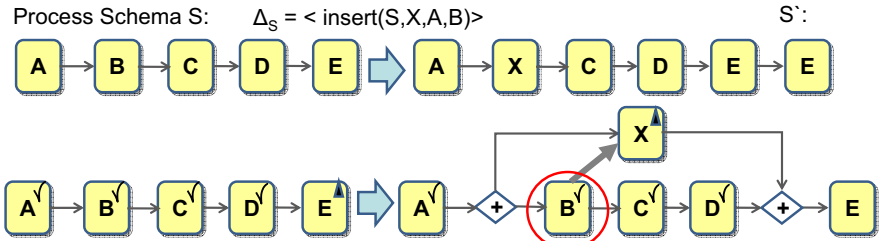

Fig. 11. Illustration of Lemmata 1 and 2

Lemma 2 (Structural Distance for Instance-Specific Adjustments (2)). *Let the assumptions be as defined for Theorem 1. Then for* $\overline{\Delta_I}$ *= move(S',X,Y,C) with* $Y \in succ^*(S',A)$ *(with* $S'[\overline{\Delta_I} > \overline{S_I}$*, I compliant with* $\overline{S_I}$*):*

$sD(\overline{S_I},S') = |succ^*(S', A) \cap pred^*(S', C)| - 1 + |succ^*(S', A) \cap pred^*(S', Y)|$

Proof Sketch of Lemma 2:

The basic consideration behind Lemma 1 is that change $\overline{\Delta_I(S')}$ can be "mapped" to change $\Delta_I(S')$ as described in Theorem 1. This can be done as follows: Consider Fig. 11: X can be inserted between A and E; i.e., by change $\Delta_I(S')$ = move(S',X,A,E). According to Lemma 1 for each activity X is moved parallel to, X "looses" one successor. Shifting the left anchor from A towards E can be expressed by inserting a sync edge between new anchor Y and X; e.g., if we chose B as left anchor (i.e., $\overline{\Delta_I(S')}$ = move(S',X,B,E)). By doing so, B "receives" new successor X. If, for example, D is chosen as left anchor, B, C, and D have new successor X. Thus, for all activities "between" A and Y one new successor occurs. □

Proof of Theorem 1:
Preconditions:

1. Δ_S = <insert(S,X,A,B)>
2. S[$\Delta_S > S'$
3. $I \in \mathcal{I}$ on S
4. $NS_I(B)$ ∈{Running, Completed}
5. $\Delta_I(S')$ = move(S',X,A,C) with
   C ∈ nextNonStartedSucc(S',I,B) $\Longrightarrow$ $NS_I(C)$ ∈ {NotActivated, Activated}
6. S'[$\Delta_I(S') > S_I$

Statement:
(1) I compliant with S_I
∧
(2) dS(S',S_I) = min{dS(S',$\tilde{S_I}$) | $S'[\tilde{\Delta_I(S')} > \tilde{S_I}$ with I compliant with $\tilde{S_I}$}

We proof Theorem 1 by contradiction.

Contradicton Assumption:

¬((1) ∧ (2)) ≡ ¬(1) ∨ ¬ (2) ≡
¬ ((1) I compliant with S_I ∧
  (2) dS(S',S_I) = min{dS(S',$\tilde{S_I}$) | $S'[\tilde{\Delta_I(S')} > \tilde{S_I}$
    with I compliant with $\tilde{S_I}$}) ≡
(¬1) I is not compliant with S_I ∨
  (¬2) $\exists\overline{\Delta_I(S')}$ with dS(S',S_I) ≥ dS(S', $\overline{S_I}$) with
    S'[$\overline{\Delta_I(S')} > \overline{S_I}$

Preliminaries for Case (¬1): Basically, compliance is defined based on execution traces [2, 7]; i.e., instance I is compliant with changed type schema version S' if trace σ_I^S of I on S could have been also produced on S'. As discussed in [7], for example, checking compliance based on generating and replaying execution traces can be quite expensive. Thus, comparable to serializability in database

systems, for example, we have stated precise *compliance conditions* (cf. [7, 8, 12, 20], based on which compliance can be quickly checked. The idea is to exploit the semantics of the applied change operation (e.g., an insert or move operation) and to evaluate the node states NS_I of "critical" activities; i.e., activities which constitute the *change region* of the the particular activity. For deletion, for example, the change region is built by the activity to be deleted (change semantics). It can be shown that compliance is ensured if the activity to be deleted is in state NotActivated or Activated. In the context of Theorem 1 we are interested in move operations.

The compliance conditions for move operation move(S',X,A,C) are as follows:
$NS_I(X) \in$ {NotActivated, Activated} $\wedge$ $NS_I(C) \notin$ {Running, Completed}

Case ¬(1): I is not compliant with $S_I \Longrightarrow$
$\Delta_I(S')$ violates the compliance condition for move operations, i.e.,
$NS_I(X) \notin$ {NotActivated, Activated}
$\vee$ $NS_I(C) \notin$ {NotActivated, Activated} $\Longrightarrow$ Contradiction $\lightning$

since $NS_I(X)$ = NotActivated since X is newly inserted at type schema level and $NS_I(C) \in$ {NotActivated, Activated} according to Precondition (5)

Case ¬(2): $\exists\overline{\Delta_I(S')}$ with dS(S',S_I) $\geq$ dS(S', $\overline{S_I}$) with S'$[\overline{\Delta_I(S')} > \overline{S_I}$
Pre-Considerations:
We assume implicitly that dS(S',S_I) and dS(S', $\overline{S_I}$) are defined $\Longrightarrow$
N' = $N'_I = \overline{N_I}$
for S'= (N', CtrlE, ...), S_I = (N_I, CtrlE$_I$, ...), and $\overline{S_I}$ = ($\overline{N_I}$, $\overline{CtrlE_I}$, ...).
From this assumption we can conclude that opType($\overline{\Delta_I(S')}$) = move (cf. Table 1)
Since we assume further that I is compliant with $\overline{S_I}$, subject($(\overline{\Delta_I(S')}$) = S follows (cf. Table 1); i.e., altogether $\overline{\Delta_I(S')}$ = move(S',X,Y,Z).
Consequently, $\Delta_I(S')$ and $\overline{\Delta_I(S')}$ differ in their parameter list. Here we can distinguish the following cases:

Case A: Y = A, Z $\neq$ C; i.e., maintaining "left anchor" A
With Lemma 1 we obtain:
sD($\overline{S_I}$,S') = $|succ^*(S', A) \cap pred^*(S', Y)| - 1 \overset{!}{<} |succ^*(S', A) \cap pred*(S', C)|$-1
Under the assumption that $\overline{\Delta_I}$ is a valid change (i.e., Y $\in$ succ*($\overline{S_I}$,A)) and compliance conditions hold for $\overline{\Delta_I(S')}$ (i.e., $NS_I(Y) \notin$ {Running, Completed}) a contradiction to the preconditions results, since in this case C cannot be one of the next non-started successor of A in S' $\lightning$

Case B: Y $\neq$ A, Z = C; i.e., maintaining "right anchor" C
Case B1: Y $\in$ succ*(S',A)
With Lemma 1 and Lemma 2 we obtain:
sD($\overline{S_I}$,S') = $|succ^*(S', A) \cap pred^*(S', C)|$-1 + $|succ^*(S', A) \cap pred^*(S', Y)| \overset{!}{<}$
$|succ^*(S', A) \cap succ^*(S', C)|$-1 $\Longrightarrow$ Contradiction $\lightning$
Case B2: Y $\in$ pred*(S',A)

Intuitively clear. If we move X between a predecessor of A and C, X is moved parallel to A. Thus at least A "looses" X as successor and we obtain $\mathrm{sD}(\overline{S_I},\mathrm{S'}) \geq \mathrm{sD}(S_I,\mathrm{S'}) + 1$

Case C: $\mathrm{Y} \neq \mathrm{A}$, $\mathrm{Z} \neq \Longrightarrow$ follows from Case A and Case B $\square$

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*
Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm