Flexibility in Process-aware Information Systems

Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam

Institute of Databases and Information Systems, Ulm University, Germany,
{manfred.reichert, stefanie.rinderle, peter.dadam}@uni-ulm.de

Abstract. Process-aware information systems (PAIS) must be able to deal with uncertainty, exceptional situations, and environmental changes. Needed business agility is often hindered by the lacking flexibility of existing PAIS. Once a process is implemented, its logic cannot be adapted or refined anymore. This often leads to rigid behavior or gaps between real-world processes and implemented ones. In response to this drawback, adaptive PAIS have emerged, which allow to dynamically adapt or evolve the structure of process models under execution. This paper deals with fundamental challenges related to structural process changes, discusses how existing approaches deal with them, and shows how the various problems have been exterminated in ADEPT2 change framework. We also survey existing approaches fostering flexible process support.

1 Introduction

In many application domains process-aware information systems (PAIS) will be not accepted by users if rigidity comes with them [1–4]. Instead, it should be possible to quickly implement new processes, to enable on-the-fly adaptations of running ones, to defer decisions regarding the exact process logic to runtime, and to evolve implemented processes over time. Consequently, process flexibility has been identified as one of the fundamental needs for any PAIS and different enabling technologies have emerged [5–8]. They support adaptive processes [9–11], declarative models [7], late modeling [12, 13], and data-driven processes [14, 15]. Basically, we need to be able to deal with uncertainty, to cope with exceptions, and to evolve processes over time:

- *Ability to deal with uncertainty.* The implemented process is based on a loosely or partially specified model, where the full specification is made during runtime and may be unique to each process instance. Rather than enforcing control through a rigid, or highly prescriptive model, that attempts to capture every aspect, the model is defined in a more declarative or incomplete way that allows individual instances to determine their own processes.
- *Ability to adapt processes.* The implemented process is able to react to exceptions, which may or may not be foreseen and which affect one or a few instances. Generally, it must be possible to adapt the structure and/or state of the process model of a particular instance. Respective adaptations, however, must not affect other instances being executed on this model as well.

– *Ability to evolve processes.* A process model has to be changed when the business process evolves. One challenge concerns the handling of long-running, active instances, which were initiated based on the old model, but now need to comply with the new specification. Potentially, thousands of active instances may be affected.

This paper focuses on structural adaptations of process models at different levels. Adaptations of single process instances (e.g., to add, delete or move activities) become necessary to deal with exceptional situations and often have to be accomplished in an ad-hoc manner [11]. Model changes at the process type level, in turn, have to be continuously conducted to evolve the PAIS [9, 5]. It must be also possible to dynamically migrate running process instances to new model versions. Important challenges are to perform instance migrations on-the-fly, to guarantee compliance of migrated instances with the new model version, and to avoid performance penalties. Our ADEPT2 change framework addresses these challenges and explicitly covers the latter two kinds of flexibility; i.e., the adaptation and evolution of processes. However, through its ability to support late binding of sub-processes and to dynamically evolve or define these sub-processes, ADEPT2 is also able to support late modeling, and thus to deal with certain kinds of uncertainty.

The ultimate ambition of structural process adaptations during runtime is to ensure correctness of the modified instances afterwards. First, structural and behavioral soundness have to be guaranteed already at the model level (i.e., without considering instance states). Second, when performing instance adaptations this must not lead to flaws (e.g., deadlocks); i.e., none of the guarantees ensured by formal checks at build time must be violated due to the runtime adaptation. As example consider Fig. 1 where the model on the left-hand side is structurally modified by arranging parallel activities `B` and `C` in sequence afterwards. The instance running on the old model (with `B` being enabled and `C` being completed) does not comply with the new model version since its marking cannot be transferred to it (`B` must be completed before `C` may start). Such undesired runtime situations are denoted as *dynamic change bug* [16]. To exterminate them adequate correctness criteria are needed; e.g., to decide whether a given process instance is compliant with a modified process model and – if yes – how to adapt instance states when migrating the instance to the new model version.

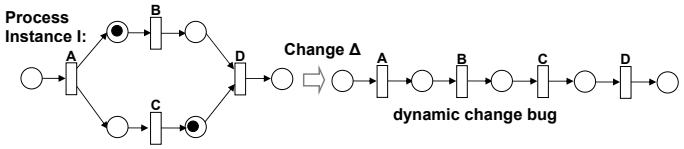

Fig. 1. Dynamic change bug

In the following we deal with different correctness notions for dynamic process changes and discuss the strengths and weaknesses of the approaches relying on

them. Based on these considerations we show how we deal with respective issues in ADEPT2, which constitutes one of the very few adaptive PAIS which allows for structural process changes during runtime at instance and type level. Section 2 introduces fundamental challenges emerging in the context of dynamic process changes and discusses existing approaches for dealing with them. Section 3 shows how ADEPT2 tackles the different challenges and exterminates dynamic change bugs. This includes both the control and the data flow perspective as well has the viewpoint of users. We survey alternative solutions for process flexibility in Section 4 and conclude with a summary in Section 5.

2 Fundamental challenges of dynamic process changes

2.1 Basic notions

When implementing a new process in a PAIS its logic has to be explicitly defined based on the provided *process meta model*. For each *business process* to be supported, a *process type* represented by a *process schema* (i.e., *process model*) is defined. For one particular process type several schemes may exist representing the different versions and the evolution of this type over time. Based on a process schema an arbitrary number of new *process instances* can be created and executed. The PAIS orchestrates them according to the defined process logic.

For defining structural process adaptations two options exist. On the one hand, respective schema adaptations can be defined based on a set of change primitives (e.g., to add or delete edges). Following this approach, realization of a particular structural adaptation usually requires the application of multiple change primitives. To specify structural adaptations at this low level of abstraction, however, is a complex and error-prone task. Another, more favorable option is to base structural adaptations on high-level change patterns [6, 17], which abstract from the concrete schema transformations to be conducted (e.g., to add a process fragment parallel to an activity or to move a fragment to a new position). Instead of specifying a set of change primitives the user applies one or few high-level change operations to define the required structural 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 $i = 1, \ldots, n,\ n \in \mathbb{N}$ sequentially. Then:*

1. *S[Δ> S' if and only if Δ is correctly applicable to S. S' is the process schema resulting from the application of Δ to S (i.e., S' $\equiv$ S + Δ). We call a change Δ correctly applicable to a schema S if all formal pre-conditions of Δ are met for S or resulting schema S' is a correct process schema according to the correctness criteria set out by the process meta model of interest.*
2. *S[Δ>S' if and only if there are process schemas $S_1, S_2, \ldots, S_{n+1} \in \mathcal{PS}$ with $S = S_1$, S' = S_{n+1} and for $1 \leq i \leq n$: $S_i[\Delta_i > S_{i+1}$ with $\Delta_i = <op_i>$*

We assume that change Δ is applied to a *sound* process schema S [18]; i.e., S obeys the specific correctness constraints set out by the used process meta model

(e.g., bipartite graph structure for Petri Nets). We denote this as *structural soundness*. We further claim that S' must obey *behavioral soundness*; i.e., any instance executed on S' must not run into a deadlock or livelock. This can be achieved in two ways. Either Δ itself preserves soundness based on pre-/post-conditions of the applied change patterns [11], or Δ is first applied on a schema copy and soundness of the resulting schema version S' is checked afterwards.

Another basic notion used in the following is *process trace*. Such trace sequentially logs the entries about the start and completion of process activities.

Definition 2 (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 $e_i \in \{Start(a), End(a)\}$, $a \in \mathcal{A}$, $i = 1, \ldots, k,\ k \in \mathbb{N}$). The temporal order of e_i in σ_I^S reflects the order in which activities were started and/or completed over S.*[1]

2.2 Under which conditions may process instances be adapted?

Most approaches dealing with structural instance adaptations [16, 19, 10, 5, 8] focus on correctness; i.e., applying a change to a running instance must neither violate its structural nor behavioral soundness. The correctness criteria used by adaptive PAIS vary and have led to different implementations [9]. Basically, there are *structural* and *behavioral* correctness criteria. While criteria from the former group try to structurally relate the process schema before the change to the resulting schema version [16, 8] (e.g., using inheritance relations for realizing the schema mapping), the latter are based on execution traces; i.e., they compare which traces are producible on a process schema before and after its change.

Structural criteria. One approach relying on *structural criteria* in connection with dynamic changes exists for *WF Nets* [16]. A WF Net is a labeled place/transition net representing a control flow schema [16, 20]. A *sound* WF Net has to be connected, safe, and deadlock free as well as free of dead transitions. Furthermore, sound WF Nets always properly terminate. Behavior of a process instance is described by a *marked WF net*. Core idea of the corresponding change framework is as follows: An instance I on schema S (represented by a marked WF Net) is considered as *compliant* with the modified schema $S' := S + \Delta$, if S and S' are related to each other under given *inheritance relations*; i.e., either S is a subclass of S' or vice versa. The following two kinds of inheritance relations are used [16]: A schema S is a subclass of another schema S' if one cannot distinguish behaviors of S and S' anymore either (1) when only executing activities of S which also belong to S' or (2) when arbitrary activities of S are executed, but only effects of activities being present in S' as well are taken into account. Thus, Inheritance Relation (1) works by *blocking* and Inheritance Relation (2) can be

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

realized by *hiding* a subset of the activities from S. More precisely, *blocked* activities are not considered for execution. *Hiding* activities implies that they are renamed to the silent activity τ. (A silent activity τ has no visible effects and is used, for example, for structuring purposes.) Consider the example from Fig. 2 where the newly inserted activities `X` and `Y` are hidden by labeling them to the silent activity τ. Thus, S' is a subclass of S. Further inheritance relations can be obtained by combined hiding and blocking of activities. Based on these inheritance relations we can state the following correctness criterion:

Criterion CC 1 (Compliance under inheritance relations) *Let S be a process schema which is correctly transformed into another schema S' by applying change Δ. Then instance I on S is compliant with S' if S and S' are related to each other under inheritance (see [16] for a formal definition).*

CC 1 ensures structural and behavioral soundness of instance I after applying change Δ to it. The question remains how to ensure CC 1; i.e., how to check whether Δ is an inheritance preserving change and therefore S and S' are related under inheritance. [16] defines precise *conditions* with respect to S and S'. When inserting a new net N into S, S and S' will be related under inheritance if N and S have exactly one place in common. This will be the case, for example, if a cyclic structure N_c is inserted into S (resulting in S') as shown in Fig. 2. Since S' is a subclass of S when hiding `X` and `Y` in N_c, soundness of I on S' is guaranteed. Checking inheritance of arbitrary process schemes is PSPACE-complete [16].

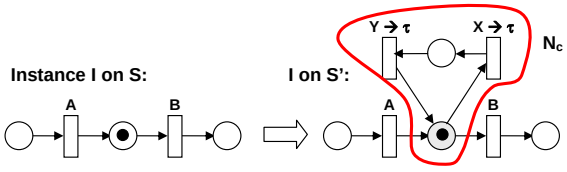

Fig. 2. Inheritance preserving change: insertion of cyclic structure

Using inheritance relations restricts the set of applicable changes to additive and subtractive ones. There is no adequate relation based on hiding/blocking activities in connection with order-changing operations. Nevertheless, this approach covers many relevant changes and copes with them without need for accessing instance states. It can be used for both correctness checks on single instances and on instance collections (e.g., WF Nets with colored tokens). It is debatable whether it also works with *concurrent changes*. Assume, for example, that instance I on S is changed resulting in instance-specific schema S_I, which is related to S under inheritance. Assume further that at process type level S is changed to S' (which is again under inheritance with S). Then it has to be analyzed whether S_I and S' are also related to each other under inheritance.

Behavioral criteria. A widely-used correctness property is the trace-based *compliance criterion* introduced by [19]. Intuitively, change Δ on schema S (i.e., $S[\Delta > S']$) can be correctly applied to instance I on S iff the execution of

I, taken place so far, can be "simulated" on the new schema version S' as well. [19] bases compliance on *trying to replay trace* σ_I^S of I on S'. If this is possible, behavioral soundness can be guaranteed when migrating I to S' [19, 5]. In summary, compliance is fundamental for changing both, single instances and instance collections. Basically, it also allows for concurrent changes. We discuss respective extensions in Section 3.5. Finally, the idea of preserving traces by structural changes based on Petri Nets is described in [21, 10].

2.3 How to adapt instance states after dynamic changes?

In addition to decide whether change Δ can be correctly applied to an instance, it becomes necessary to properly and correctly adapt instance states afterwards.

Structural approaches. [16] provides *transfer rules* based on the aforementioned inheritance relations (cf. Criterion CC 1) to cope with marking adaptations in the context of WF net changes. After applying change Δ to schema S (i.e., $S[\Delta > S'$), necessary marking adaptations are realized by mapping markings of instances running on S onto markings on S'. Adapting markings after inserting parallel branches, for example, is complicated since in some cases we have to insert additional tokens to avoid deadlocks. Fig. 3 shows an example. By just mapping the token of s_3 on S to place s_3 on S', a deadlock is produced. [16] proposes to insert an additional token on s_5 (*progressive transfer rule*). Though the resulting marking on S' is correct, the semantics of newly inserted tokens is debatable, particularly, if colored tokens (i.e., data flows) are considered as well.

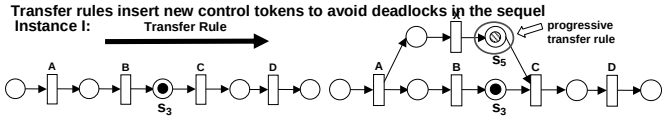

Fig. 3. Marking adaptation policy in [16, 20]

Another approach has been proposed for Flow Nets [10] for which an explicit mapping between the markings of the net before and after the change has to be specified. This is done manually by adding *flow jumpers*; i.e., transitions mapping tokens from the old to the new net (cf. Fig. 4). Both single instances or instance collections can be migrated. The handling of concurrent changes at instance and type level, however, is cumbersome, since several new net versions have to be merged with the old net via flow jumpers. Manually specifying mappings between instance markings is not a realistic option in practice. As it can be see from Fig. 4 respective mappings already become complex for simple scenarios.

Behavioral approaches. Checking compliance means to replay instance traces on the changed process schema. Thus, marking adaptations come for free. However, at the presence of thousands of running instances, replaying whole traces becomes too expensive. In Sect. 3 we introduce a more sophisticated approach for automatically checking compliance and adapting instance markings. It

has been realized in ADEPT2 [5] and utilizes specific properties of the ADEPT2 meta model as well as the semantics of the ADEPT2 change patterns.

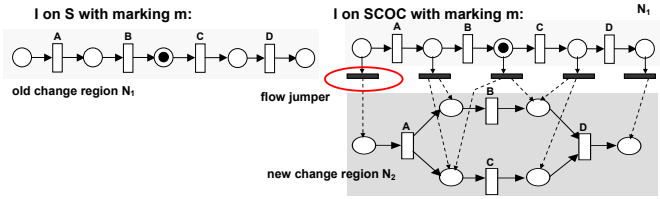

Fig. 4. Marking mapping (Synthetic Cut Over Change) [10]

2.4 Discussion

Generally, a correctness criterion is needed which preserves structural and behavioral soundness of the dynamically adapted instances (cf. Fig. 5). This criterion should be valid independent from the used process meta model. Nonetheless, it is always applied in the context of a concrete meta model and change framework. Like serializability in database systems, defining a proper correctness notion is only one side of the coin. The other is to check it efficiently, particularly at the presence of a multitude of instances. When applying the criterion for a particular meta model, logical optimizations for checking it can be based on exploiting meta model properties as well as the semantics of the applied change operations. Additional optimizations are conceivable at the implementation level.

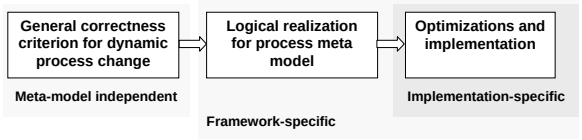

Fig. 5. Correctness of process change – general view

3 Dynamic process changes in ADEPT2

We now elaborate compliance as meta model independent correctness criterion in the context of a concrete process meta model (i.e., ADEPT2 WSM Nets [5]). We show how compliance can be efficiently checked and instance markings be automatically adapted when performing dynamic instance changes.

3.1 WSM Nets

Well-Structured Marking-Nets (WSM Nets) as applied in ADEPT2 can be used to represent process schemes by attributed serial-parallel graphs (cf. Fig. 6a). Consider Fig. 6a, which depicts an example of a WSM Net. A WSM Net S

is structurally sound if the following constraints hold: S has a unique *start* and a unique *end node*. Except for these start and end nodes each activity node has at least one incoming and one outgoing *control edge* $e \in CtrlE$[2]. Structuring nodes such as AND-Splits, XOR-Split, AND-Joins, and XOR-Joins can be distinguished based on their node type (6a). Loop backs can be explicitly modeled via loop edges $e \in LoopE$ (cf. Fig. 6a). Basically, WSM Nets are *block-structured*, where control blocks (sequences, branchings, loops) can be nested, but must not overlap. We additionally allow to relax this block structure and to synchronize the execution order of activities from parallel branches by means of so-called *sync links* $e \in SyncE$ if required. Such sync links must not cross the boundary of a loop block; i.e., an activity from a loop block must not be connected with an activity from outside the loop block via a sync link (and vice versa). Furthermore, $S_{fwd} = (N, CtrlE, SyncE)$ constitutes an acyclic graph which allows to exclude deadlocks due to cyclic "wait-for" dependencies.

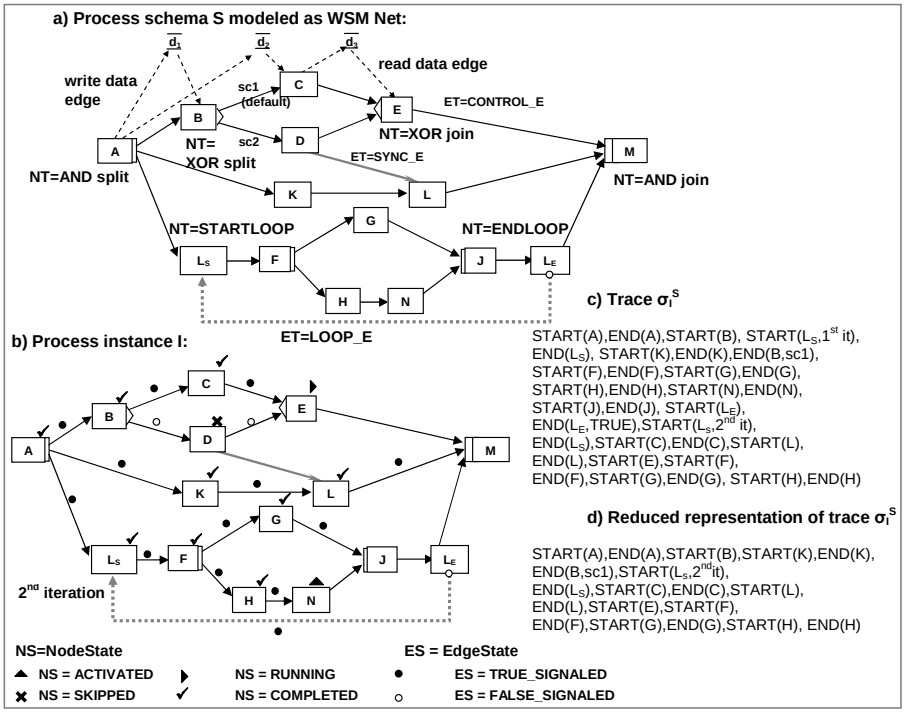

Fig. 6. WSM Net with running instance, traces, and marking rules

[2] i.e.; S is connected

For WSM Nets, data flow is realized by associating process data elements to activities by read and write edges (cf. Fig. 6a). For activities with mandatory input parameters linked to global data elements, it has to be ensured that respective data elements are always written by a preceding activity at runtime independent of which execution path is chosen.

Taking the WSM Net S from Fig. 6a new process instances can be created and executed (cf. Fig. 6b). Thereby, the execution state of an instance I is captured by marking function $\mathrm{M}^{S_I} = (\mathrm{NS}^{S_I}, \mathrm{ES}^{S_I})$ where S_I denotes the instance-specific schema of I. M^{S_I} assigns to each activity n its current status $NS(n)$ and to each edge e its current marking $ES(e)$. Markings are determined according to well defined firing rules. Based on the local context of an activity (i.e., incoming and outgoing edges), the activity marking can be determined [11]; markings of already passed regions and skipped branches are preserved (except loop backs). Activities marked as `Activated` are ready to fire (i.e., enabled) and can be worked on. Their status then changes to `Running` and afterwards to `Completed`. Activities belonging to non-selected execution branches obtain marking `Skipped` and can no longer be selected for execution (e.g., activity D in Fig. 6). Concerning data elements, different versions of a data object can be stored, which is important for the handling of partial rollback operations.

To cope with exceptional situations, instances can be individually modified by applying high-level change patterns (e.g., to insert or move activities). For such individually modified instances the *instance-specific* schema deviates from the original one they were started on. Respective instances are denoted as *biased*. To capture information about instance-specific changes, logically, each instance I runs on an instance-specific schema S_I with $S[\Delta_I > S_I$; Δ_I denotes the instance-specific *bias*. For unbiased instances, $\Delta_I = <>$ and consequently $S_I \equiv S$ hold. According to the change patterns framework presented in [6, 22], Tab. 1 presents some *high-level change operations*, which can be used to define or structurally modify process schemes. A high-level change operation realizes a particular variant of a change pattern (e.g., serial or parallel insertion of activities). In ADEPT2 these change operations include formal pre- and post-conditions. They automatically perform necessary schema transformations while ensuring structural soundness. One typical example of such a change operation is the insertion of an activity and its embedding into the process context.

Currently, we are working on an extension of the ADEPT2 meta model to further increase expressiveness and to cover frequent workflow patterns (see [23] for details). Generally, there exists a trade-off between expressiveness of a meta model and support for structural adaptations in imperative approaches. ADEPT2 has been designed with the goal to enable the latter, i.e., to allow for the efficient implementation of adaptation patterns, restrictions on the process meta model are made. Similar restrictions in terms of expressiveness hold for other approaches supporting structural adaptations [24, 8]. On the other hand, YAWL is a reference implementation for workflow patterns and therefore allows for a high degree of expressiveness [25]. Structural adaptations have not yet been

Table 1. *A selection of high-level change operations on process schemas*

Change pattern	Design choice	Effects on schema S
AP1: Insert activity		
	serial	inserts the activity between directly succeeding ones
	insert between node sets	
	without condition	inserts the activity parallel to existing ones
	with condition	conditional insert of the activity
AP2: Delete activity		deletes the activity from schema S
AP3: Move activity		
	serial	moves the activity to position between directly succeeding activities
	move between node sets	
	without condition	moves the activity parallel to existing ones
	with condition	conditional move of the activity

addressed in YAWL and their implementation would be more difficult due to the higher expressiveness (see Section 4 for more details).

3.2 Checking compliance in ADEPT2

In Section 2 two approaches for ensuring correctness of dynamically adapted instances are presented. CC 1 enables correctness checks for process changes without taking instance state into account. However, this comes for the price of a restricted set of change patterns (e.g., no order-changing operations). On the other side, traditional compliance [19] uses full instance information as captured by execution traces. Doing so allows for all kinds of change patterns. However, traditional compliance has turned out to be too restrictive (e.g., in conjunction with loops). Apart from this it is expensive to check. ADEPT2 follows an elegant compromise between these two compliance criteria abolishing their particular limitations. This is achieved by extending traditional compliance to overcome its restrictiveness. Furthermore, precise conditions for ensuring compliance are elaborated, which only take dedicated instance information into account. First of all, we formalize traditional compliance criterion CC 2:

Criterion CC 2 (Compliance of unbiased instances) *Let S be a sound process schema and let I be an unbiased process instance running on S with associated execution trace* σ_I^S*. Assume that change* Δ *transforms S into another sound process schema* S' *(i.e.,* $S[\Delta > S'$*). Then: I will be compliant with S' (i.e., it can migrate to S') if its execution trace* σ_I^S *can be correctly replayed on S'.*

CC 2 depends on the representation (i.e. *view*) of trace σ_I^S. One is the `Start/End` view on σ_I^S. It logs both start and end events of executed activities (cf. Fig. 6c). Taking this view on σ_I^S we obtain an instance with correct marking when replaying it on S' [9]; i.e., I can continue execution based on S' afterwards while structural and behavioral soundness are preserved. However, this view is too restrictive in conjunction with changes of cyclic process structures [5]. If a loop is affected by a change, but has already undergone some

iterations, the respective instance will be always considered as non-compliant with S' (i.e., trace entries related to finished iterations cannot be replayed on the adapted schema) though a migration of this instance would not lead to errors in the sequel. ADEPT2 therefore applies a *reduced representation* $\sigma^S_{I\,red}$ of σ^S_I, which corresponds to a (logical) projection of σ^S_I only on current loop iterations; i.e., for loop activities we only consider entries written during the last iteration of the respective loop (cf. Fig. 6d). Note that this approach is fostered by the block-structuring of WSM Nets. In addition, *data flow correctness* can be ensured by enriching execution traces with information about data access; i.e., read and write access on data elements. This is crucial in connection with dynamic process changes [5]. We dig into data flow correctness in Section 3.4.

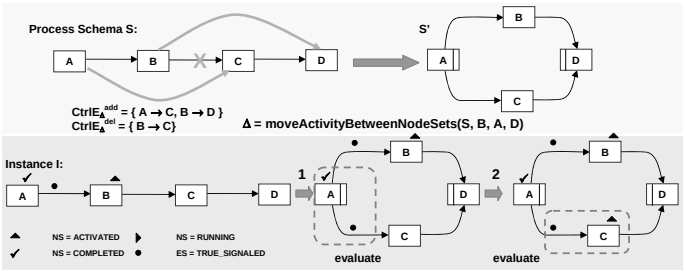

Fig. 7. Adapting markings for WSM Nets

CC 2 constitutes a logical correctness notion similar to serializability in database systems. Another challenge is to efficiently check it. A naive solution would be to try to replay instance traces on S' and to verify whether resulting instance states on S' are correct. Obviously, this can cause a performance penalty if a multitude of instances shall be migrated. Generally, a change framework has to provide methods which ensure CC 2 and can be efficiently checked. ADEPT2 provides methods which make use of the semantics of the applied change operations (cf. Tab. 1) and the model-inherent markings of WSM Nets for all change patterns supported [5, 6]. Contrary to many approaches (e.g., [26, 16]), ADEPT2 is able to deal with order-changing operations as well. (A discussion on the complexity of compliance checking for different change patterns and a comparison with other approaches can be found in [9].) As example consider Fig. 7 where activity B is moved to the position between activities A and D. Instead of replaying complete trace σ^S_I of I on S', according to the ADEPT2 *compliance conditions* for moving activities, the following has to be checked: I is compliant with S' if for all newly inserted control edges in $CtrlE^{add}_{\Delta}$ their destination activities are not running or completed yet. In the latter case (i.e., state of respective activities is `Running` or `Completed`), the state of the associated source node is `Completed` and compliance can be only ensured if the entries of source and destination node within trace $\sigma^S_{I\,red}$ have the right order (i.e., `END` entry of source node before `START` entry of destination node). For our example from Fig.

7, the destination activities of edges in $CtrlE_{\Delta}^{add}$ (i.e., C and D) have not been started yet. Consequently, activity B can be moved as described for instance I.

As can be seen from this example, moving activities is one of the few cases, where we might have to exploit additional information from trace $\sigma_{I\,red}^{S}$. In connection with newly added control edges, the associated orders must be already reflected by the entries of the trace. If the destination activities of the new control edges have not been started yet, the right order will be always guaranteed. Otherwise, the actual order has to be checked based on the execution trace. For inserting and deleting activities, checking node states is sufficient (see [27, 28] for a complete summary of compliance conditions and respective proofs).

3.3 Adapting instance markings in ADEPT2

We have described how CC 2 can be ensured and which information is needed. Our goal was to prevent access to whole instance traces. By holding this maxim we now discuss how compliant instances can be automatically migrated to an adapted schema. One challenge, not adequately solved by other approaches, constitutes the efficient and correct adaptation of instance markings. According to CC 2, the marking of a migrated instance must be the same as it can be obtained when replaying its (reduced) trace on the new schema version. How extensive marking adaptations turn out depends on the kind and scope of the change. Except from initialization of newly added nodes and edges, no marking adaptations become necessary if the instance has not yet entered the change region. In other cases more extensive marking adaptations are required. An activated activity `X`, for example, will have to be deactivated if control edges are inserted with `X` as target activity. Conversely, a newly added activity will have to be activated or skipped if all predecessors already have marking `COMPLETED` or `SKIPPED`.

We utilize information on the change context to decide on marking adaptations. We illustrate this by means of an example. Consider Fig. 7 where B is moved to the position between A and D. The algorithm first determines which nodes and edges have to be potentially (re)marked. In the given case these sets can be determined based on the inserted and deleted control edges. Then, the algorithm steps through the initial node and edge sets and adapts instance markings step by step. In Fig. 7, these steps are denoted as intermediate steps. First of all, for newly inserted control edge $A \longrightarrow C$, an adaptation has to be done; since source node A is already completed, $A \longrightarrow C$ is marked as `True_Signaled`. Consequently, in the next step, destination node C has to be marked as `Activated` since all incoming edges have marking `True_Signaled`. For the other newly inserted edge $B \longrightarrow D$ and deleted edge $A \longrightarrow B$ no marking adaptation becomes necessary. Thus, the algorithm terminates with the desired marking of I on S'.

Based on the compliance criterion, the **dynamic change bug** as discussed in literature (e.g. [29, 16] is not present anymore in ADEPT2. More precisely, the application of the change operation as depicted in Fig. 1 would be rejected in ADEPT2 based on the corresponding compliance conditions. Furthermore, even for order-changing operations, markings can be automatically adapted without need for interacting with users. Basically, the described approach for ensuring

compliance can be transfered to other process meta models as well. We have shown this for BPEL [30] and for Activity Nets [31].

3.4 Data Flow Correctness

So far, we have not considered data flow correctness in connection with process changes. Basically, we have to ensure correctness of the modeled data flow when directly changing it (e.g., by adding or deleting data edges) as well as when adapting the associated control flow structure. Regarding the latter, ADEPT2 will only allow for control flow changes if data flow correctness can be preserved afterwards [28]. As example take process schema S from Fig. 7 and assume that B writes data element d and C reads it afterwards. Regarding this scenario, data flow correctness would be not preserved if we conducted the depicted adaptation (i.e., to move B from its position between A and C to the position parallel to C). Since B would then be ordered in parallel to C, we could not guarantee any longer that B writes d before C reads this data element. As another scenario, assume that change Δ inserts two activities A and B in an arbitrary schema S, where A is writing data element d, which is read by B afterwards. In this case, Δ would not be correctly applicable, if A is inserted within one branch of an alternative branching. In this case, it cannot be ensured that A is activated during runtime and d is written accordingly.

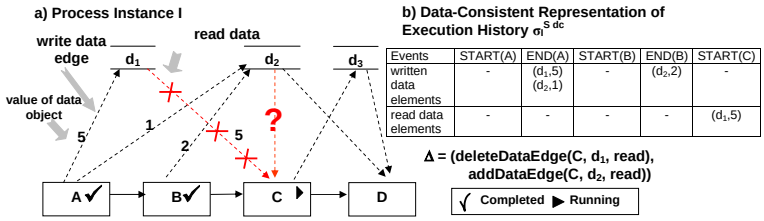

Fig. 8. Data Consistency Problem

Altogether, to avoid such structural flaws, a given sequence of change operations can be only applied in ADEPT2 if structural and behavioral soundness is guaranteed afterwards. In the given example, the structural pre-conditions of the *move* operation would disallow the application of the intended change since in schema in S' the read access of C to d has no preceding write access to this data element.

Another challenge is to preserve the correctness of the data flow schema when changing it. As example consider the scenario depicted in Fig. 8. Activity C has been started and has already read value 5 of data element d_1. Assume that, due to a modeling error, read data edge $(C, d_1, read)$ shall be deleted and read data edge $(C, d_2, read)$ be inserted instead. Consequently, C should have read value 2 of data element d_2 (instead of data value 5). Such inconsistent read behavior has to be prohibited since it can lead to errors and inconsistencies in the sequel (e.g., if instance execution is aborted and therefore has to be rolled back). Using

any representation of execution trace σ_I^S as introduced so far, this erroneous case cannot be detected, i.e., the the instance would be classified as compliant.

We need an adapted form of σ_I^S considering data flow as well. We denote $\sigma_I^{S^{dc}}$ as data-consistent trace representation of σ_I^S with $\sigma_I^{S^{dc}} = <e_1, \ldots, e_k>$: $e_i \in \{\text{START(a)}^{(d_1,v_1),\ldots,(d_n,v_n)}\ \text{END(a)}^{(d_1,v_1),\ldots,(d_m,v_m)}\}$, 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 $(i = 0, \ldots, k)$. Using this data-consistent representation of σ_I^S the problem illustrated in Fig. 8a is resolved.

3.5 Concurrent process adaptations

Being able to cope with changes of single instances or a collection of instances in isolated manner is crucial to meet practical needs. However, changes do not always occur separately from each other. Assume that instance I on schema S is modified due to an exception resulting in instance-specific schema S_I (i.e. $S[\Delta_I > S_I$). If later S is changed as well due to new legal regulations resulting in S' (i.e. $S[\Delta > S'$), the challenge is to decide how to cope with *concurrent* changes Δ_I and Δ (cf. Fig. 9): May I migrate to S' and - if yes - how does the instance-specific change (i.e., bias) turn out on S'? The latter question is particularly interesting if Δ_I and Δ are *overlapping*; i.e., they have some or all change effects in common (e.g., deletion of same activity). Then Δ_I has to be adapted on S' since S' already reflects parts of Δ_I. Fig. 9 illustrates the different cases in connection with dynamic change. If S is transformed into S', the user might want to exclude some of the instances due to specific constraints. For all others, migration to S' is desired. First, we have to distinguish between instances still running according to S (unbiased instances) and those individually modified (biased instances). For biased instances it is further important to know whether concurrent schema and instances changes are *disjoint* or *overlap* since further migration strategy depends on that. For all instances we need adequate correctness criteria (see [28, 32] for respective extensions of compliance and migration strategies).

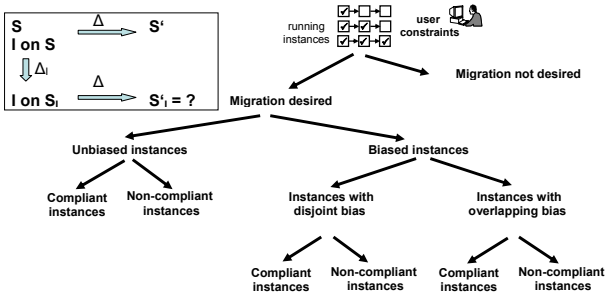

Fig. 9. Instance migration – big picture

3.6 How users interact with ADEPT2?

So far, ADEPT2 has been applied in several domains including healthcare, automotive development, construction engineering, logistics, and e-negotiation [1, 2, 33, 34]. While for some applications the provided ADEPT2 clients were sufficient to adequately assist users in adapting their processes [34, 1], in other cases specific client components were implemented based on the application programming interfaces offered by ADEPT2. AgentWork [33], for example, provides a rule-based planning component for the healthcare domain that automatically derives adaptations of patient treatment processes to be applied in a given context. Here, users only have to approve the suggested instance changes, which are then automatically carried out by the system; i.e., ADEPT2 serves as engine to implement the changes. CONSENSUS [1], in turn, uses the existing ADEPT2 clients to realize the flexibility and dynamism needed to accommodate to the various contingencies and obstacles that can appear during e-negotiations.

In all these case studies the provision of high-level change patterns and the change framework described were considered as strong points in favor of ADEPT2. Based on the lessons learned we are currently extending the meta model for WSM nets with additional workflow patterns [23]. Furthermore, we developed techniques targeting at improved user assistance. In [35], for example, we present an approach which uses conversational case-based reasoning to allow for the reuse of previously applied ad-hoc changes in similar problem context. We are also developing mechanisms to incorporate semantical constraints into adaptive PAIS in order to prohibit semantically counterproductive changes [36]. ADEPT2 expresses semantic constraints in terms of rules and verifies them during buildtime, runtime, and in connection with process changes. We further provide an authorization component, which allows to restrict process changes to authorized users, but without nullifying the advantages of a flexible PAIS by handling authorizations in a too rigid way [37]. Finally, we are investigating the concept of process views in connection with dynamic process changes [38]. Basic idea is to provide abstract views to users and to allow them to apply changes to these views and to propagate the view updates back to the underlying process.

4 Discussion

To effectively deal with exceptions through structural process adaptations and to enable process evolution have been major design goals of the ADEPT2 technology. In the previous sections we have presented basic issues and concepts to attain these goals and to enable dynamic structural changes of different process aspects. This section provides a survey on the state-of-the art (see also [9, 17]), but extends it with a summary of approaches dealing with uncertainty as well. Furthermore we discuss alternative solutions for enabling process flexibility including declarative approaches [7] and case handling [14]. For a discussion of techniques for process evolution we refer to [9, 17].

Dealing with Exceptions. While *expected exceptions* are usually considered during buildtime by specifying exception handlers to resolve the respec-

tive exceptions during runtime [39], non-anticipated situations, in turn, may require structural adaptations of single process instances [3, 11]. A comprehensive overview of exception handling mechanisms is provided by [40]. Depending on the type of exception different handling strategies can be pursued (e.g., to roll back parts of the process), which are described as *exception handling patterns* in [40]. Exception handling often requires combined use of such patterns resulting in rather complex routines. The Exlet approach [41], for example, addresses this problem by allowing for the combination of different exception handling patterns to an exception handling process called Exlet. Similarly, [42] suggests the usage of meta workflows for coordinating exception handling. While exception handling patterns are well suited for dealing with expected exceptions, non-anticipated situations, in turn, often require *structural adaptations* of individual process instances as well [39]. Besides ADEPT2, several other approaches support ad-hoc changes [8, 24, 6], however, only the ADEPT2 framework allows for high-level change patterns (e.g., to insert, move or delete activities and process fragments, respectively) instead of change primitives (e.g., to add or delete nodes and edges in the process graph) [6]. To ensure correctness of run-time changes, soundness needs to be guaranteed. When conducting instance-specific changes, using change primitive (e.g., WASA2 [8] or CAKE2 [24]), soundness of the resulting process schema cannot be guaranteed and correctness of a process schema has to be explicitly checked after applying the respective set of primitives. ADEPT2, in turn, associates pre-/ post-conditions with the high-level change patterns, which allows to guarantee soundness. Finally, PAISs supporting instance-specific adaptations should be able to cope with concurrent changes as well. While many system prohibit concurrent process instance changes (e.g., FLOWer [14], WASA2 [8]), ADEPT2 supports them based on optimistic concurrent change techniques; CAKE2, in turn, supports concurrent process instance changes using pessimistic locking [24].

Dealing with Uncertainty. Flexible PAIS must be also able to cope with *uncertainty*. Common to existing approaches is the idea to defer decisions regarding the exact control-flow to runtime [13, 17]. Instead of requiring a process model to be fully specified prior to execution, parts of the model can remain unspecified and be refined during run-time when more information is available. Examples for such techniques are Late Binding, Late Modeling and Late Composition of Process Fragments. Finally, data-driven processes provide for some flexibility regarding the exact control-flow as well [15, 14, 43].

Late binding allows to defer the selection of activity implementations to runtime; i.e., the implementation of the respective activity is chosen out of a set of process fragments at runtime either based on rules or user decisions [17]. As example consider Worklets [13], which allow for late binding of sub-process fragments to activities. At buildtime, the respective activity is modeled as a placeholder. *Late Modeling and Composition*, in turn, are techniques which go one step beyond by allowing parts or whole of the process to be defined during runtime [12, 44]. Late Modeling allows for modeling selected parts of a process

schema at runtime. At buildtime a placeholder activity as well as constraints for modeling the respective sub-process are defined. Usually, the modeling of the placeholder activity needs to be completed before its execution can start. Even more flexibility is provided by Late Composition. It allows users to compose existing process fragments on-the-fly; e.g., by dynamically introducing control dependencies between them. There is no predefined schema, but the (sub-)process instance is created in an ad-hoc way by selecting from the available fragments and obeying the predefined constraints. For both techniques, the model being dynamically defined may or may not be controlled by constraints. Complete lack of constraints can defeat the purpose of a PAIS, where as too many constraints may introduce rigidity that compromises the dynamic process [45].

[46, 12] propose an approach for the late modeling of process fragments. A part of the process (termed *Pocket of Flexibility*) is deemed to be of a dynamic nature and is defined through a set of activities and a set of constraints defined on them. At runtime, the undefined part is detailed for a given process instance based on tacit knowledge and obeying the prescribed constraints. In contrast, the approach provided by DECLARE [44, 7] enables late composition of process fragments. Basically, the whole process is defined in a declarative way. However, DECLARE can also be used in combination with imperative languages (e.g., YAWL). In this scenario, not the entire process model is described in a declarative way, but only sub-processes. Like in the *Pocket of Flexibility* approach a process model is defined as a set of activities and a set of constraints. During runtime process instances can be composed whereby any behavior is allowed which is not prohibited by any constraints. *Data-driven processes* as supported by the *case handling* tool FLOWer [14] do not predefine the exact control-flow, but orchestrate the execution of activities based on the data assigned to a case. Thereby, different kinds of data objects are distinguished. Mandatory and restricted data objects are explicitly linked to one or more activities. If a data object is mandatory, a value will have to be assigned to it before the activity can be completed. If a data object is restricted for an activity, this activity needs to be active in order to assign a value to the data object. Free data objects, in turn, are not explicitly associated with a particular activity and can be changed at any time during a case execution and consequently provide for flexibility during run-time. [47] compares workflow management and case handling with means of a controlled experiment. Recently, additional paradigms for the data-driven modeling and adaptation of large process structures have emerged. In particular, they allow for the transformation of data model changes to process adaptations as well as for sophisticated exception handling procedures [48, 15].

5 Summary and Outlook

We have provided a general discussion on flexibility issues in adaptive PAIS and we surveyed the state-of-the-art. As core of any approach enabling dynamic process changes, adequate correctness notions are needed. When implementing them within a PAIS and making use of the formal properties of the underlying

process meta model as well as change framework, different optimizations can be realized. Similarly, optimized techniques for the automated adaptation of instance states can be provided when migrating process instances to a modified schema. Along these challenges, we have discussed different correctness criteria and their application to specific process meta models. On one side we have considered structural criteria and their (logical) realization within Petri-net based PAIS. On the other side, we have analyzed approaches using traces for deciding whether an instance is compliant with a modified schema. Since both kinds of approaches come along with limitations, we have presented the ADEPT2 approach. ADEPT2 uses consolidated instance data and exploits the semantics of the applied change operations in order to abolish the limitations of pure structural and behavioral approaches. Finally, we have addressed issues related to concurrent changes, data flow correctness, and use of ADEPT2. Future work will extend our analysis of correctness criteria for dynamic process change. We will elaborate to what degree existing correctness notions can be relaxed to increase the number of compliant process instances [32]. Furthermore, there are still many open questions regarding the realization of concurrent process changes (e.g., how to deal with partly overlapping changes) and the management of the process variants resulting from instance changes. In this context, we are developing intelligent analysis techniques to learn from process changes [49–51]. Finally, we are currently working on issues related to the dynamic adaptation of organizational rules and access constraints [52, 53], to process variant management [54], and to process model refactoring [55].

References

1. Bassil, S., Keller, R., Kropf, P.: A workflow–oriented system architecture for the management of container transportation. In: BPM'04. (2004) 116–131
2. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Data and Knowledge Engineering **61** (2007) 39–58
3. Müller, R., Greiner, U., Rahm, E.: AgentWork: A workflow system supporting rule–based workflow adaptation. Data and Knowlege Engineering **51** (2004) 223–256
4. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In: BPM'06. LNCS 4102 (2006) 368–377
5. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116
6. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: CAiSE'07. LNCS 4495 (2007) 574–588
7. Pesic, M., Schonenberg, H., Sidorova, N., van der Aalst, W.: Constraint-based workflow models: Change made easy. In: CoopIs'07. (2007) 77–94
8. Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects. University of Münster (2000) Habilitation Thesis.
9. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering **50** (2004) 9–34
10. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COOCS'95. (1995) 10–21

11. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems **10** (1998) 93–129
12. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specifications. In: ER'01. (2001) 513–526
13. Adams, M., Hofstede, A., Edmond, D., van der Aalst, W.: Worklets: a service-oriented implementation of dynamic flexibility in workflows. In: CoopIS'06. (2006) 291–308
14. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data and Knowledge Engineering **53** (2005) 129–162
15. Müller, D., Reichert, M., Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In: CAiSE'08. LNCS 5074 (2008) 48–63
16. van der Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science **270** (2002) 125–203
17. 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
18. Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: BPM'05. (2005) 386–391
19. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering **24** (1998) 211–238
20. van der Aalst, W., Weske, M., Wirtz, G.: Advanced topics in workflow management. Int'l Journal of Integrated Design and Process Science **7** (2003)
21. Haddad, S., Pradat-Peyre, J.: New efficient petri nets reductions for parallel programs verification. Parallel Processing Letters **16** (2006) 101–116
22. Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In: ER'08. LNCS 5231, Barcelona (2008) 279–293
23. Wolz, J.: New control and data flow concepts in ADEPT2. Master's thesis, Ulm University (2008)
24. Minor, M., Schmalen, D., Koldehoff, A., Bergmann, R.: Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. In: WETICE'07. (2007)
25. van der Aalst, W., ter Hofstede, A.: Yawl: Yet another workflow language. Information Systems **30** (2005) 245–275
26. Agostini, A., De Michelis, G.: Improving flexibility of workflow management systems. In: BPM'00. (2000) 218–234
27. Rinderle, S., Reichert, M., Dadam, P.: Supporting workflow schema evolution by efficient compliance checks. Technical Report UIB2003-02, Ulm University (2003) (available at http://www.uni-ulm.de/in/iui-dbis/forschung/publikationen.html).
28. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, Ulm University (2004)
29. van der Aalst, W.: Exterminating the dynamic change bug: A concrete approach to support worfklow change. Information Systems Frontiers **3** (2001) 297–317
30. Reichert, M., Rinderle, S.: On design principles for realizing adaptive service flows with BPEL. In: EMISA'06. (2006) 133–146
31. Reichert, M., Rinderle, S., Dadam, P.: On the common support of workflow type and instance changes under correctness constraints. In: CoopIS'03. LNCS 2888 (2003) 407–425

32. Rinderle-Ma, S., Reichert, M., Weber, B.: Relaxed compliance notions in adaptive process management systems. In: ER'08. LNCS 5231, Barcelona (2008) 232–247
33. Müller, R.: Event-Oriented Dynamic Adaptation of Workflows. PhD thesis, University of Leipzig, Germany (2002)
34. Golani, M., Gal, A.: Optimizing exception handling in workflows using process restructuring. In: BPM'06. (2006) 407–413
35. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing integrated life cycle support in process-aware information systems. Int'l Journal of Cooperative Information Systems (IJCIS) **18** (2009)
36. Ly, L., Rinderle, S., Dadam, P.: Integration and verification of semantic constraints in adaptive process management systems. DKE **64** (2008) 3–23
37. Weber, B., Reichert, M., Wild, W., Rinderle, S.: Balancing flexibility and security in adaptive process management systems. In: CooplS '05. LNCS 3760 (2005) 59–76
38. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: BPM'07. LNCS 4714 (2007) 88–95
39. Reichert, M., Dadam, P., Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Software and Systems Modeling (SOSYM) **2** (2003) 37–58
40. Russell, N., van der Aalst, W., ter Hofstede, A.: Exception Handling Patterns in Process-Aware Information Systems. In: CAiSE'06. (2006) 288–302
41. Adams, M., ter Hofstede, A., van der Aalst, W., Edmond, D.: Dynamic, extensible and context-aware exception handling for workflows. In: CoopIS'07. (2007)
42. Kumar, A., Wainer, J.: Meta workflows as a control and coordination mechanism for exception handling in workflow systems. Dec. Support Sys. **40** (2004) 85–105
43. Rinderle, S., Reichert, M.: Data-driven process control and exception handling in process management systems. In: Proc. CAiSE'06. LNCS 4001 (2006) 273–287
44. Pesic, M.: Constrained-based Workflow Management Systems – Shifting Control to Users. PhD thesis, TU Eindhoven (2008)
45. Wainer, J., de Lima Bezerra, F.: Constraint-Based Flexible Workflows. In: Groupware: Design, Implementation, and Use. Springer (2003)
46. Mangan, P., Sadiq, S.: A constraint specification approach to building flexible workflows. J of Research and Practice in Inf Technology **35** (2002) 21–39
47. Mutschler, B., Weber, B., Reichert, M.: Workflow management versus case handling: Results from a controlled software experiment. In: SAC'08. (2008) 82–89
48. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: Proc. CoopIS'07. LNCS 4803 (2007) 131–149
49. Li, C., , Reichert, M., Wombacher, A.: Discovering reference process models by mining process variants. In: ICWS'07, Beijing (2008) 45–53
50. Guenther, C., Rinderle-Ma, S., Reichert, M., van der Aalst, W., Recker, J.: Using process mining to learn from process changes in evolutionary systems. Int'l Journal of Business Process Integration and Management **3** (2008) 61–78
51. Li, C., Reichert, M., Wombacher, A.: On measuring process model similarity based on high-level change operations. In: ER'08. LNCS 5231, Barcelona (2008) 248–264
52. Rinderle-Ma, S., Reichert, M.: Managing the life cycle of access rules in CEOSIS. In: Proc. EDOC'08, Munich (2008) 257–266
53. Rinderle-Ma, S., Reichert, M.: A formal framework for adaptive access control models. In: Journal of Data Semantics, IX. LNCS 4601 (2007) 82–112
54. Hallerbach, A., Bauer, T., Reichert, M.: Managing process variants in the process lifecycle. In: ICEIS'08, Barcelona (2008) 154–161
55. Weber, B., Reichert, M.: Refactoring process models in large process repositories. In: CAiSE'08. LNCS 5074, Montpellier (2008) 124–139