Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 50 (2004) 9–34
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Correctness criteria for dynamic changes in workflow systems—a survey ☆

Stefanie Rinderle *, Manfred Reichert, Peter Dadam

Department Databases and Information Systems, Faculty of Computer Science, University of Ulm, James-Franck-Ring, 89069 Ulm, Germany

Available online 25 January 2004

Abstract

The capability to dynamically adapt in-progress workflows (WF) is an essential requirement for any workflow management system (WfMS). This fact has been recognized by the WF community for a long time and different approaches in the area of adaptive workflows have been developed so far. This survey systematically classifies these approaches and discusses their strengths and limitations along typical problems related to dynamic WF change. Along this classification we present important criteria for the correct adaptation of running workflows and analyze how actual approaches satisfy them. Furthermore, we provide a detailed comparison of these approaches and sketch important further issues related to dynamic change.

Keywords: Workflow management; Adaptive systems; Dynamic workflow changes; Correctness criteria

1. Introduction

A rapidly changing environment and a turbulent market force any company to change their business processes ever more frequently [1]. Process changes become necessary, for example, when new laws come into effect, optimized or restructured business processes are to be implemented, exceptional situations occur, or reactions to a changed market are required. Therefore, a critical

☆ This work was done within the research project "Change management in adaptive workflow management systems", which is funded by the German Research Community (DFG).
* Corresponding author. Tel.: +49-731-50-24229; fax: +49-731-50-24134.
E-mail addresses: rinderle@informatik.uni-ulm.de (S. Rinderle), reichert@informatik.uni-ulm.de (M. Reichert), dadam@informatik.uni-ulm.de (P. Dadam).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2004.01.002

challenge for the competitiveness of any enterprise is its ability to quickly react to business process changes and to adequately deal with them [16,31].

As pointed out in [17,22,28,34], basically, WF changes can take place at two levels—the *WF type* and the *WF instance level*. *Instance-specific changes* are often applied in an ad-hoc manner and become necessary in conjunction with real-world exceptions. They usually affect only *single WF instances*. As opposed to this, in conjunction with *WF schema changes* at the WF type level, a collection of related instances may have to be adapted. There are many approaches supporting such *adaptive* workflows [1,5,9,18,21,26,28,34]. All of them present very interesting, but partially strongly differing ideas and solutions. Therefore, it is an important job to summarize central correctness criteria for adaptive workflows and to compare actual approaches along them. In this survey, we focus on three fundamental issues regarding dynamic WF changes:

(1) *Completeness*. Users must not be unnecessarily restricted, neither by the applied WF meta model nor the offered change operations. Therefore, expressive control/data flow constructs must be provided [7]. For practical purposes, at minimum, change operations for inserting and deleting activities as well as control/data dependencies between them are needed.
(2) *Correctness*. The ultimate ambition of any adaptive WF approach must be correctness of dynamic changes [1,5,9,18,21,26,28,34]. More precisely, we need adequate *correctness criteria* to check whether a WF instance I is *compliant* with a changed WF schema or not; i.e., whether the respective change can be correctly *propagated* to I without causing inconsistencies or errors (like deadlocks or improperly invoked activity programs). These criteria must not be too restrictive, i.e., no WF instance should be needlessly excluded from being adapted to a process change.
(3) *Change realization*. Assuming that a dynamic change can be correctly propagated to an instance I (along the stated correctness criteria), it should be possible to automatically *migrate* I to the new schema. In this context, one challenge is to correctly and efficiently adapt instance states.

In the following, we provide a classification of actual approaches based on the operational semantics of the underlying WF meta models and on the kind of correctness criteria applied for dynamic WF changes (Sections 2 and 3). Section 3 introduces a selection of typical *dynamic change problems* and discusses strengths and weaknesses of the approaches when dealing with these problems. A detailed comparison of the different approaches is presented in Section 4. We sketch important change scenarios and existing approaches in Section 5 and close with a summary in Section 6.

2. Workflow meta models of adaptive workflow approaches

Current approaches supporting adaptive workflows are based on different WF meta models. Very often, the solutions offered by them are dependent on the expressiveness as well as on the formal and operational semantics of the used formalism. Fig. 1 summarizes WF meta models for which adaptive WF solutions have been realized. According to [15] we classify those meta models with respect to their operational semantics and the evaluation strategies applied for executing WF instances during runtime. The first strategy uses only one type of (control flow) token passing

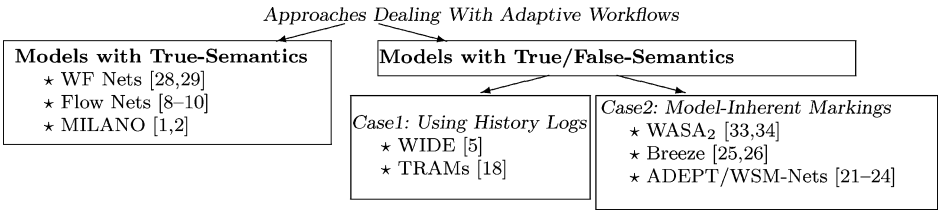

Fig. 1. Meta models of approaches supporting adaptive workflow.

through each WF instance (*True-Tokens*). The other strategy is based on two types of tokens—*True-* and *False-Tokens*. (An early approach for True-/False-Tokens in bipolar synchronization schemes was presented in [11].) Simplistically, True-Tokens trigger activities that are to be executed next and False-Tokens describe skipped activities. Formalisms which solely use True-Tokens include, for example, Petri-Nets [1,9,28] (cf. Section 2.1). Approaches which, in addition, use False-Tokens to represent skipped activities or skipped execution branches can be found in the area of graph-/activity-based meta models [5,18,21,26,34] (cf. Section 2.2). They can be further divided according to the way they represent the tokens. One possibility is to gain them from *execution histories* [5], which log events like activity start and completion. Alternatively, special (*model-inherent*) activity markings, which represent a consolidated view on the history logs, can be used [21,26,34].

In the following, for each approach shown in Fig. 1, we sketch the basic formalism used for WF modeling and execution together with its structural and dynamic properties. This background information is useful for better understanding the criteria applied by these approaches to guarantee dynamic change correctness (cf. Section 3).

2.1. Approaches with true-semantics

WF Nets. A *WF Net* is a labeled place/transition net $N = (P, T, F, l)$ representing a control flow schema [28,29]. Thereby, P denotes the set of places, T the set of transitions, $F \subseteq (T \times P) \cup (P \times T)$ the set of directed arcs, and l the labeling function, which assigns a label to each transition. Data flow issues are excluded. A WF Net must have one initial place i and one final place f. In [28] 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, i.e., the end state—which contains one token in f and no other tokens—is always reachable. The behavior of a WF instance is described by a *marked WF net* (N, m) with marking function m and associated firing rules. A transition t is enabled if each of its input places contains a token. If t fires, all tokens from its input places are removed and to each output place of t a token is added.

Flow Nets. The operational semantics of Flow Nets [8–10] is comparable to (safe) WF Nets but with one major difference: Places can be equipped with more than one token. Chautauqua [10] offers an implementation where Flow Nets are generalized to *Information Control Networks* (*ICN*). An ICN bases WF enactment on instance-specific data tokens. Different WF instances are distinguished by coloured tokens and are controlled by the same ICN.

MILANO Nets. Another Petri-Net-based approach is offered by MILANO [1,2]. As opposed to WF Nets and Flow Nets the expressiveness of MILANO Nets is restricted to marked, acyclic

Free-Choice Petri Nets (so called *Net Models* (NM)). Data flow is not explicitly considered. An NM S can be mapped to a *Sequential Model* (SM) which represents global states and state transitions of S. Thus, SM corresponds to the *reachability graph* of S.

Interestingly, the above approaches abstract from internal activity states, i.e., they only differentiate between activated and non-activated transitions. As we will see later, this coarse classification may be unfavorable in conjunction with certain kind of dynamic changes.

2.2. Approaches with true/false-semantics

As opposed to the above approaches, the following WF meta models distinguish between different states an activity may go through. Generally, initial status of an activity is set to `NotActivated`. It changes to `Activated` when all preconditions are met. Activity execution is then either started automatically or corresponding worklist entries are generated. When starting activity execution its status changes to `Running`. Finally, at successful termination, status passes to `Completed`. In addition, some of the models assign status `Skipped` to activities belonging to non-selected execution branches. Usually, in addition, an execution history $\Pi_I^S = \langle e_0, \ldots, e_k \rangle$ is maintained for each instance I with $e_i \in \{(\mathtt{S}_a, \langle \mathtt{var}, \mathtt{val} \rangle^*),\ (\mathtt{E}_a, \langle \mathtt{var}, \mathtt{val} \rangle^*)\}$. For each started activity X the values of process data elements read by X and for each completed activity Y the values of data elements written by Y are logged.

2.2.1. Case 1: approaches based on history logs

WIDE Graphs: WIDE [5] uses an activity-based WF meta model which allows the modeling of sequential, parallel, conditional, and iterative activity executions. A WF schema has to meet several constraints to be correct: first there must be a path from the start activity to all other activities and the end activity has to be reachable from all of them. The other constraints refer to the correct use of splits and joins. Furthermore, each WF schema S is associated with a set of global process variables whose values may be read or written by activity instances during runtime. A particular instance I is described by its schema S and its execution history Π_I^S. As opposed to the following approaches, WIDE only logs activity completion events.

TRAMs Graphs. In TRAMs [18]—in contrast to other graph-/activity-based approaches—control flow is not realized by control edges. Instead, it is described in a declarative way by using conditions for starting/finishing activities. WF schema correctness is preserved by invariants, i.e., schema-related conditions which must be fulfilled. Data flow is explicitly specified by connecting output and input parameters of subsequent activities. TRAMs distinguishes between activity states `Activated`, `Running` and `Completed`, and logs status changes in the execution history Π_I^S of the respective instance I.

2.2.2. Case 2: approaches using model inherent markings

$WASA_2$ Activity Nets. WASA_2 [33,34] uses an activity-based meta model. A *WF schema* $S = (V_S, C_S, D_S)$ is a tuple with sets of activity nodes V_S, control connectors $C_S \subset V_S \times V_S$, and data connectors $D_S \subset V_S \times V_S$. Similar to TRAMs, the flow of data is modeled by connectors which map output and input parameters of subsequent activities. A WF schema S is correct iff all input parameters are correctly mapped onto a type-conform output parameter and the graph structure is acyclic, i.e., loops are excluded. A *WF instance I* is described by an instance graph

$I = (V_I, C_I, D_I)$ whose state is denoted by model-inherent activity markings. WASA_2 distinguishes between markings `NotActivated`, `Activated`, `Running`, `Completed`, and `Skipped`.

Breeze Activity Nets. Similar to TRAMs and WASA_2, Breeze [25,26] uses model-inherent activity markings. A WF schema is described by a directed acyclic graph $W = \langle N, F \rangle$ with finite set of activity nodes N and flow relation $F \subseteq N \times N$. It is possible to model sequences, parallel/conditional branches and complex activities. WF data is described by a set of WF variables. A schema is correct if there is a unique initial node n_{i} and a unique final node n_{f}, and for all $n \in N$ there is a path from n_{i} to n_{f} via n. Breeze also ensures correct data provision of invoked activities.

ADEPT WSM-Nets. Another approach with model-inherent markings is offered by *Well-Structured Marking-Nets* (*WSM-Nets*) as applied in *ADEPT* [21]. WF schemes are represented by serial-parallel graphs $S = (N, D, \mathit{NodeType}, \mathit{CtrlE}, \mathit{SyncE}, \mathit{LoopE}, \mathit{DataE}, \mathit{EC})$ with activity set N, data element set D, and distinguishable node/edge types. Branchings of different type and loops are modeled in a block-oriented fashion. However, this restrictive structure can be relaxed by the use of *sync edges* (*SyncE*), which allow to define precedence relations between activities of parallel branches. Data flow is modeled by connecting *global process variables* (D) with activities (N) either by *read* or *write data edges* (*DataE*). A schema is correct iff $S_{\text{fwd}} = (N, \mathit{CtrlE}, \mathit{SyncE})$ is an acyclic graph, i.e., the use of sync edges must not lead to deadlock-causing cycles [21]. An instance I is defined by a tuple $(S, NS^S, ES^S, \mathit{Val}^S, \Pi_I^S)$ where S denotes the corresponding schema and $NS^S : N \mapsto \{\texttt{NotActivated}, \texttt{Activated}, \texttt{Running}, \texttt{Completed}, \texttt{Skipped}\}$ possible node states. Val^S is a function on D. It reflects for each data element $d \in D$ either its current value or `UNDEFINED` (if d has not been written yet). Finally, Π_I^S is the execution history of I.

Comparing the above formalisms we can find many differences. For example, only WF Nets, Flow Nets, WIDE Graphs, and ADEPT WSM-Nets allow the modeling of loops. Data flow issues are factored out by WF Nets and MILANO Nets. While WIDE only logs end entries of activities, the execution histories in TRAMs and ADEPT store the start entries of activities as well.

3. Classification and dynamic change correctness

In this section, we present a classification of the approaches introduced in Section 2. It is based on the *correctness criteria* applied in connection with dynamic WF changes. This classification is fundamental for better understanding the different solutions as well as their strengths and limitations. In doing so, we do not make a difference between changes of single instances and adaptations of a collection of instances (e.g., due to a WF type change). Instead we focus on fundamental correctness issues related to dynamic WF changes. In the following, let S be a WF schema and let I be an instance based on S. Assume that S is transformed into another correct schema S' by applying change Δ. What schema correctness exactly means depends on the structural and dynamic correctness properties of the used WF meta model (cf. Section 2).

3.1. Classification and problem framework

Fig. 2 presents a two-dimensional classification: The first dimension (marked on the vertical axis) is grouped by the kind of correctness criteria the different approaches are based on. The second dimension (marked on the horizontal axis) indicates on which information the different

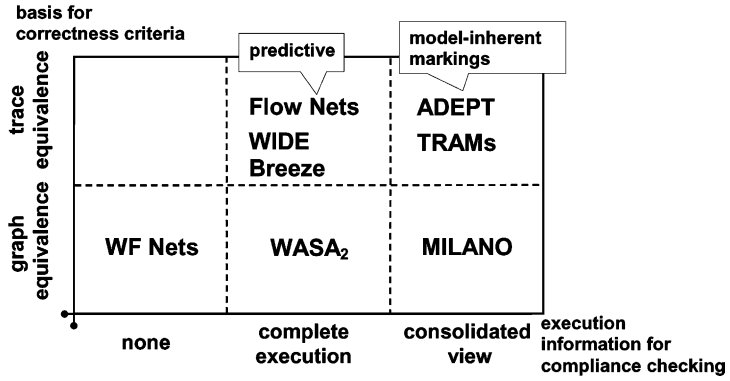

Fig. 2. Classification of approaches along the applied correctness criteria.

approaches check their particular correctness criterion. Regarding the first dimension, we distinguish between approaches founding their correctness criteria on *WF graph equivalence—WF Nets*, *MILANO Nets*, and *$WASA_2$ Activity Nets*—and approaches with correctness criteria based on *WF trace equivalence–Flow Nets*, *WIDE Graphs*, *Breeze Activity Nets*, *TRAMs Graphs* and *ADEPT WSM-Nets*. The core idea of *graph equivalence* is either to compare the respective WF schema before and after the change [28] or to map the WF instance graph of I to the changed WF schema S' [1,34]. Depending on the "degree of coverage" it can be decided whether change Δ is applicable to I or not. Generally, *trace equivalence* focuses on the work done by I so far [5,18,23,26]. If it could have been achieved on S' as well, I can be migrated to S'. A *predictive* approach is offered by Flow Nets [9] where, in addition, future instance execution on the changed schema is taken into account. Regarding the second dimension, approaches can be further distinguished depending on how they check their particular correctness criterion. Some of them consider complete history information of respective instances [5] whereas others use a consolidated view of previous instance execution [18,24].

We show how the approaches from Fig. 2 ensure correctness in conjunction with dynamic WF changes. In addition, for comparison purposes, we exemplarily discuss the approaches along five typical problems related to dynamic change (cf. Fig. 3). Due to lack of space we cannot cover all problems arising in this context. Particularly, we do not consider side-effects of control flow modifications on WF aspects other than control and data flow (e.g. temporal constraints). [1] Nevertheless, the following problems are very typical in the context of dynamic changes and therefore provide a good basis for comparing existing approaches.

(1) *Changing the Past* (*CP*). The CP problem corresponds to the rule of thumb not to "change the past of an instance". Neglecting this rule may lead to inconsistent instance states (e.g., livelocks or deadlocks) or missing input data of subsequent activity executions (see Fig. 3(1)).
(2) *Loop Tolerance* (*LT*). The LT problem refers to an approach's ability to correctly and reasonably deal with changes on loop structures (see Fig. 3(2)). In particular, approaches should not

[1] For example, there is considerable work in the literature focussing on the interplay between dynamic WF changes and the correct handling of time [20,26].

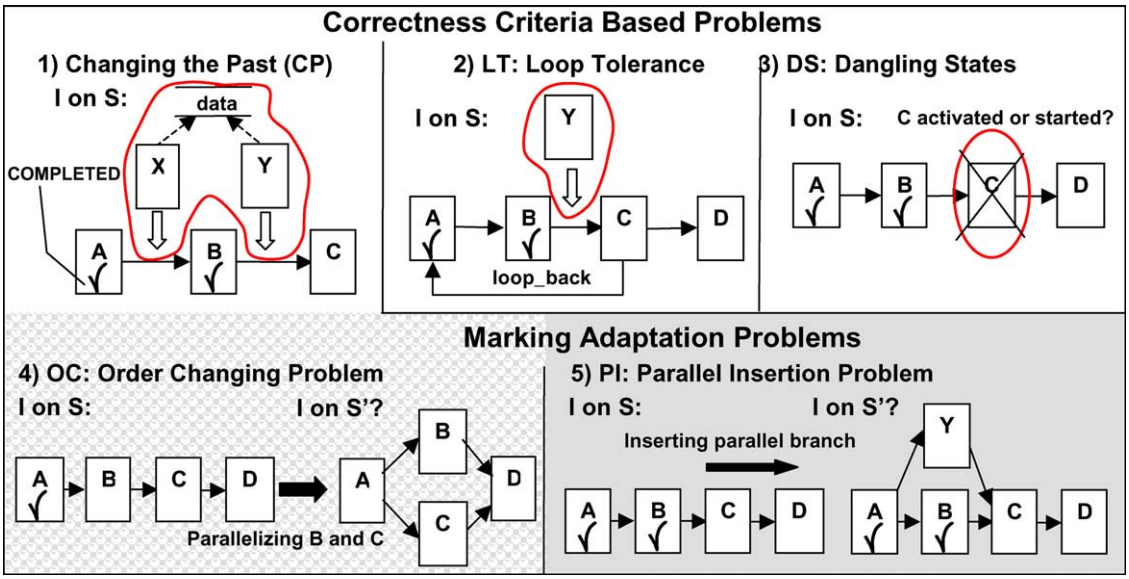

Fig. 3. Five typical problems regarding dynamic workflow change.

needlessly exclude instances from migrating to a new schema solely based on the fact that the respective changes concern loops.

(3) *Dangling States* (*DS*). The DS problem arises in conjunction with approaches not distinguishing between activated and started activities (see Fig. 3(3)). As a consequence, very often such approaches either forbid the deletion of activated activities—what is too restrictive—or they allow the deletion of already started activities—what leads, for example, to loss of work.

(4) *Order Changing* (*OC*). The OC problem refers to correctly adapting instance markings when applying order changing operations like parallelization, sequentialization, and swapping of activities (see Fig. 3(4)).

(5) *Parallel Insertion* (*PI*). As opposed to (4) the PI problem arises when inserting a new parallel branch. Concerning Petri-Nets, for example, after such a change we may have to insert additional tokens to avoid deadlocks in the sequel (see Fig. 3(5)). The OC and PI problems are closely related to the *dynamic change bug* as it has been presented in [28].

In the following we refer to these characteristic problems as the *dynamic change problems*, and we show how the different approaches supporting adaptive workflows deal with them.

3.2. Approaches based on graph equivalence

The approaches discussed in this section base their particular correctness criteria on graph equivalence [1,28,29,33,34]. Here we can further distinguish between approaches which do [1,34] and which do not use instance execution histories [28] for checking compliance.

3.2.1. Approaches not requiring instance execution information

WF Nets. The core idea of the approach presented in [28] is as follows: An instance I on schema S (represented by a marked WF Net) is 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. In this context, the following two kinds of basic inheritance relations are provided [28]: A schema S is a subclass of another schema S' if one cannot distinguish the behaviors of S and S' (1) either when only executing tasks of S which are also present in S' or (2) when arbitrary tasks of S are executed but only effects of those tasks are taken into account which are present in S' as well. Thus, inheritance relation (1) works by *blocking* and inheritance relation (2) works by *hiding* a subset of tasks of S. More precisely, *blocking* of tasks means that these tasks are not considered for execution. *Hiding* tasks implies that the tasks are renamed to the silent task τ. (A silent task τ has no visible effects and is used, for example, for structuring purposes.) One example is depicted in Fig. 5(1) where the newly inserted activities X and Y are hidden by labeling them to the silent task τ. In addition, further inheritance relations can be achieved by combining hiding and blocking of WF tasks. Based on these inheritance relations we can state the following correctness criterion.

Correctness Criterion 1 (*Compliance under inheritance relations*). Let S be a WF schema which is correctly transformed into another WF schema S'. Then instance I on S is compliant with S' if S and S' are related to each other under inheritance (for a more formal definition see [28]).

The challenging question is how to ensure Criterion 1. In [28] van der Aalst and Basten present an elegant way by providing special change operations which automatically preserve one of the four presented inheritance relations between the original and the changed schema. These change operations comprise additive and subtractive changes or, more precisely, the insertion and deletion of cyclic structures, sequences, parallel and alternative branches. Let therefore again schema S be transformed into schema S' by change Δ. In order to check whether Δ is an inheritance preserving change and therefore S and S' are related under inheritance (cf. Criterion 1) the authors define precise *conditions* with respect to S and S'.

As an example take the insertion of a cyclic structure N_c into S (resulting in S') where N_c and S have exactly one place in common (see Fig. 4). Then it can be ensured that S' is a subclass of S when hiding X and Y in N_c. Checking inheritance of arbitrary WF schemes is PSPACE-complete

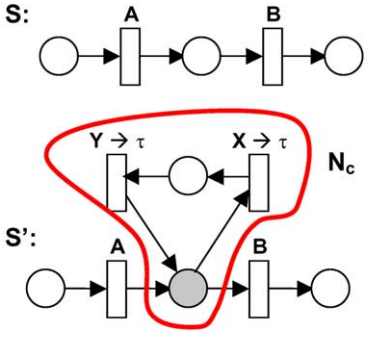

Fig. 4. Inheritance preserving change.

[28]. However, a powerful diagnosis tool called *Woflan* has been developed [28,32] to automatically decide inheritance rules for two given schemes.

As it can be seen from Fig. 5(1), using Criterion 1 it is possible to change already passed WF regions (CP problem). One problem in this context is to correctly adapt control flow tokens. However, by using anonymous tokens (i.e., excluding data tokens) the CP problem is simplified.

Using inheritance relations as described above restricts the set of applicable changes to additive and subtractive ones. More precisely, there is no adequate inheritance relation based on hiding or blocking activities when applying an order-changing operation. Consequently, the OC problem (cf. Fig. 3) is factored out. Nevertheless, van der Aalst and Basten [28] offer an original and very important contribution by ensuring compliance for many practically relevant changes without need for accessing instance data.

After having decided whether an instance I on S is compliant with S' or not (cf. Criterion 1), we need rules to adapt the marking of I on S'. For this purpose, [28] provides *transfer rules* based on inheritance relations (cf. Definition 1). After inserting activities, cyclic structures or alternative branches, necessary marking adaptations are realized by directly mapping tokens of S onto S'. The insertion of parallel branches is more complicated since in some cases we have to insert

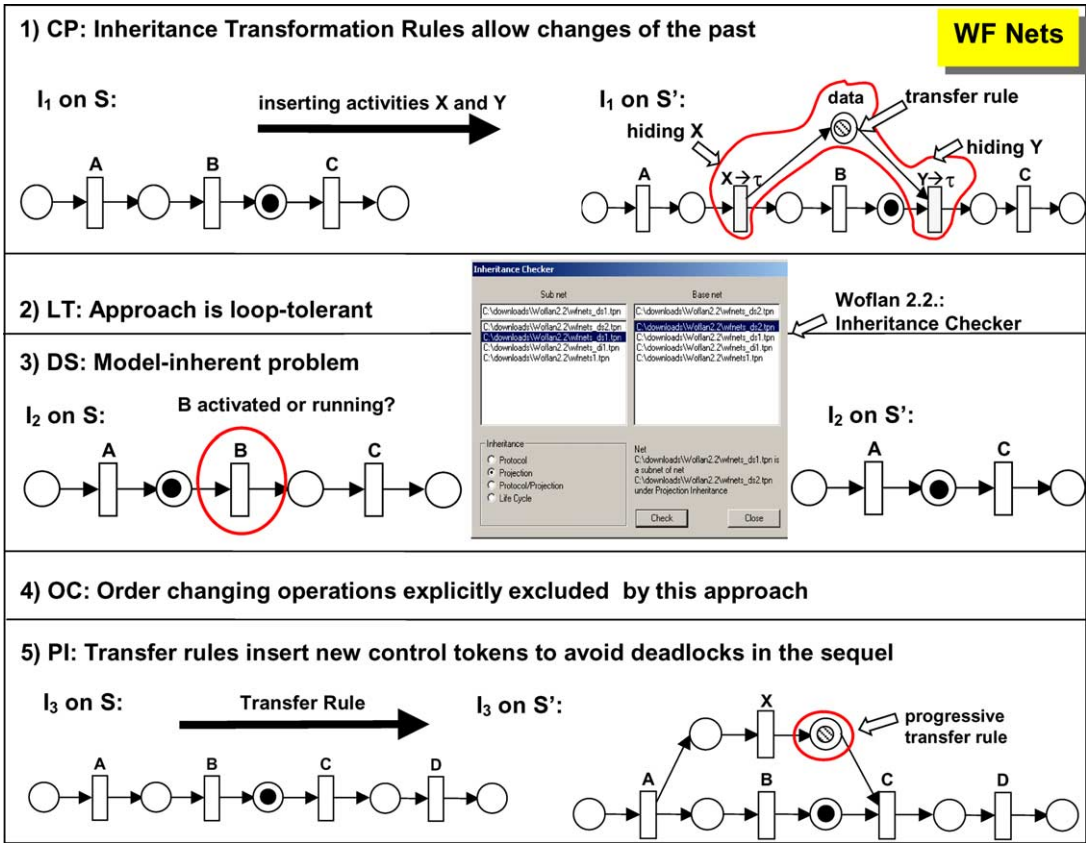

Fig. 5. Correctness checking and marking adaptations in [28,29].

additional tokens to avoid deadlocks. One example is given in Fig. 5(5) where we add one token to an input place of the parallel join transition.

3.2.2. Approaches using complete execution information

$WASA_2$ Activity Nets. $WASA_2$ [29,33,34] also uses graph equivalence to state formal correctness for dynamic changes. As opposed to WF Nets $WASA_2$ additionally takes instance information into account. More precisely, the execution state of an instance is described by its *purged instance graph* which is derived from original schema S by deleting all activities which have not been started yet and by removing all associated control and data edges.

Formally, a mapping $m : V_I \mapsto V_{S'}$ between WF instance $I = (V_I, C_I, D_I)$ and WF schema $S' = (V_{S'}, C_{S'}, D_{S'})$ assigns to every instance node $n \in V_I$ a unique schema node $m(n) \in V_{S'}$. With this, the following correctness criterion based on *valid mappings* between instance and schema graph can be stated:

Correctness Criterion 2 (*Valid mapping*). Let $I = (V_I, C_I, D_I)$ be a purged WF instance graph derived from WF schema $S = (V_S, C_S, D_S)$. Let further Δ be a change which correctly transforms S into another schema $S' = (V_{S'}, C_{S'}, D_{S'})$. Then: I is compliant with S' iff a valid mapping m: $V_I \mapsto V'_S$ exists; i.e.

$$(\forall i', j' \in V_{S'} \text{ with } \exists (i', j') \in C_{S'} \exists i,\ j \in V_I : i' = m(i),\ j' = m(j) \wedge (i, j) \in C_I) \text{ and vice versa } \wedge$$

$$(\forall k, l' \in V_{S'} \text{ with } \exists (k', l') \in D_{S'} \exists k, l \in V_I : k' = m(k),\quad {}'_l = m(l) \wedge (k, l) \in D_I) \text{ and vice versa}$$

Intuitively, an instance I can be migrated to a changed schema S' if each completed activity of I is also contained in S' and all control and data dependencies existing in I have counterparts in S' (cf. Fig. 6(5)). Criterion 2 can be paraphrased using the notion of *schema prefixes* [33] which leads to Criterion 3.

Correctness Criterion 3 (*Schema prefix*). Let $I = (V_I, C_I, D_I)$ be a purged WF instance graph derived from WF schema $S = (V_S, C_S, D_S)$. Let further Δ be a change which transforms S into another schema $S' = (V_{S'}, C_{S'}, D_{S'})$. Then: I is compliant with S' iff I is a prefix of S', i.e.,

$$V_I \subseteq V_{S'},\ C_I \subseteq C_{S'},\ D_I \subseteq D_{S'} \text{ and } \forall (p, q) \in (C_{S'} - C_I) \cup (D_{S'} - D_I) : q \notin V_I.$$

Instance graph I_4 from Fig. 6(5) is a prefix of S' but I_3 in Fig. 6(4) is not. From Fig. 6(1) we can see that Criteria 2 and 3 prohibit changes of already passed graph regions. Thus, correct data provision of activities and consistent instance states are guaranteed. Since $WASA_2$ Activity Nets are acyclic (cf. Section 2.2) the LT problem (cf. Fig. 3) is not present. Details about how Criteria 2 and 3 can be checked and instances be adapted to the changed schema have not been available. However, a powerful prototype exists. Interestingly, for some cases Criteria 2 and 3 are too restrictive regarding the OC problem (cf. Fig. 3). An example is depicted in Fig. 6(4) where I_3 could be smoothly migrated to S' but no valid mapping between I_3 and S' exists.

MILANO Nets. Only a special class of schema transformations is considered, namely parallelization, sequentialization and swapping of activities [1]. In doing so, special constraints (summarized by the *Minimal Critical Specification* (*MCS*)) are obeyed for the underlying Sequential Model (cf. Section 2). For these restricted changes the following correctness criterion is provided:

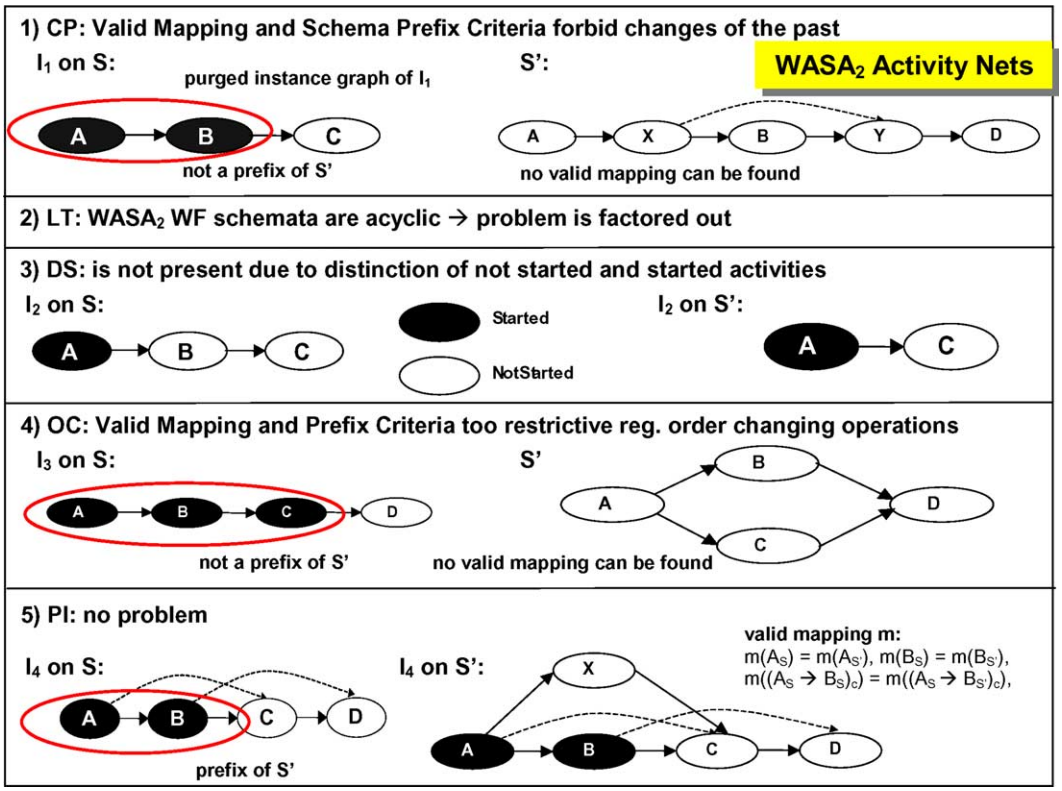

Fig. 6. Criteria 2 and 3 [33,34] applied to typical change problems.

Correctness Criterion 4 (*Safe states*). Let S be a WF schema and I an instance on S. Let further Δ be a change which transforms S into another correct WF schema S'. Then: I is compliant with S' if I is not in an unsafe state on S regarding S'. A state of S is unsafe regarding S' if this state is not present in S'.

Potential states of S and S' can be determined by constructing their Sequential Models (reachability graphs). An example for an instance with unsafe state is depicted in Fig. 7(1). For such cases MILANO postpones instance migration until the instance will be in a safe state again. Doing so cultivates the CP problem (cf. Fig. 3). The LT and the PI problem cannot be evaluated since the underlying WF schemes are acyclic and parallel insertion is not supported (cf. Section 2.1). Parallelization of activities is always allowed, but no details are given how to adapt instance markings in this context (cf. Fig. 7(5)).

3.3. Approaches based on trace equivalence

In this section we discuss approaches which base dynamic change correctness on *trace equivalence* (cf. Fig. 2).

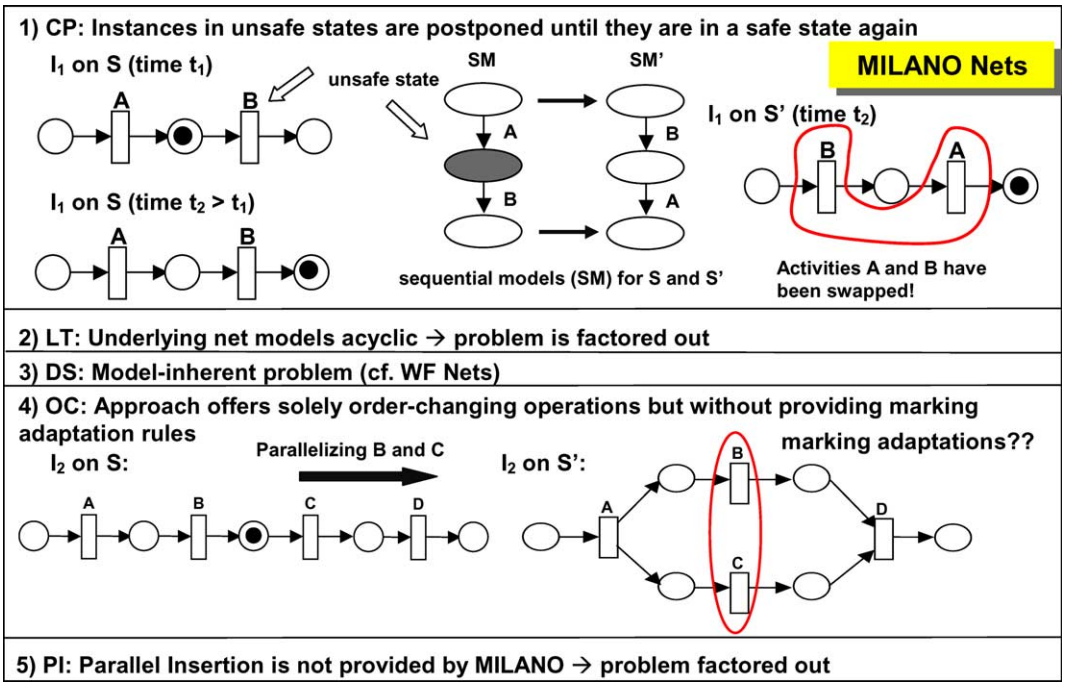

Fig. 7. Typical change problems in MILANO [1].

3.3.1. Predictive approaches

Flow Nets offer a first approach based on trace equivalence [8,9]. In [9], WF instance changes on S are carried out by substituting the marked sub-net N_1 of S, which is affected by Δ, by another marked sub-net N_2, which reflects the modifications set out by Δ. Thereby, N_1 is referred to as the *old* and N_2 as the *new change region* (cf. Fig. 8(4)). As the authors point out, the selection of the change regions cannot be fixed. Roughly, the old change region is defined as the smallest marked sub-net containing all activities affected by Δ.

For the following considerations, please remember that $\bar{i}$ denotes the initial and $\bar{f}$ the final marking of S (cf. Section 2.1). Furthermore, we formally define the *FiringSequenceSet* (*FSS*) of a schema S as follows: Let m and m' be two markings on S. Then $FSS(S, m, m')$ is the set of all possible firing sequences leading from m to m' on S.

Correctness Criterion 5 (*Pre-change firing sequence*). Let I be an instance on WF schema S with marking m and let $\omega \in FSS(S, \bar{i}, m)$. Let further Δ be a change which transforms S into another correct WF schema S' and let m' be the resulting marking of I on S'. Then I is compliant with S' iff

- $FSS(S, m, \bar{f}) \neq \emptyset \Rightarrow FSS(S', m', \bar{f}) \neq \emptyset$;
- $\forall \omega' \in FSS(S', m', \bar{f}) \Rightarrow (\omega' \in FSS(S, m, \bar{f}) \vee \omega\omega' \in FSS(S', \bar{i}, \bar{f}))$.

Criterion 5 presupposes that the marking m' resulting from the migration of instance I to the changed schema S' is known. Then starting from m' it has to be verified that all firing sequences ω'

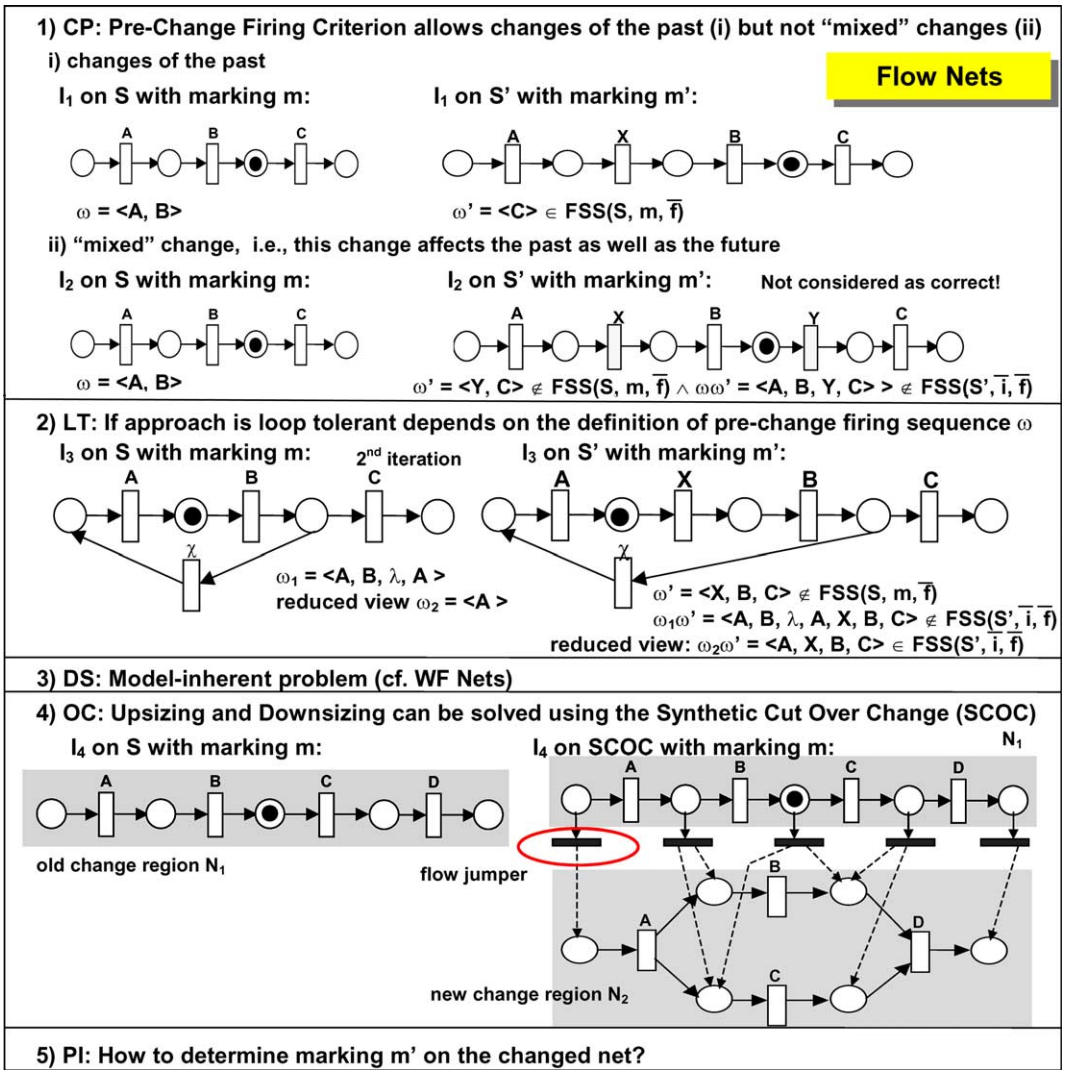

Fig. 8. Pre-change criterion and SCOC [9] applied to typical change problems.

leading from m' to the terminal marking on S' are either producible on S starting from m as well (cf. Fig. 8(1)) or firing sequence ω leading to m on S can be continued on S' by ω' (cf. Fig. 8(2)). Criterion 5 is very interesting in the context of the CP problem (cf. Fig. 3): On the one hand it allows "pure" changes of the past (cf. Fig. 8(1), i). On the other hand, it forbids changes which affect both already passed regions and regions which will be entered in the sequel (cf. Fig. 8(1), ii). Whether Criterion 5 is loop tolerant or not depends on the definition of the pre-change firing sequence ω (cf. Fig. 8(2)).

Regarding the OC problem (cf. Fig. 3) the authors present two kinds of change operations and a special change class, the *Synthetic Cut-Over Change* (SCOC). Applying SCOC, the old change region N_1 is maintained in S' together with N_2 (for an example see Fig. 8(4)); i.e., S' contains two

versions of the modified subnet. How this "fusion" of old and new change region is carried out depends on the applied change. In [9] two change scenarios—*Upsizing* and *Downsizing*—are introduced. Upsizing means that N_2 can "do more" than N_1, i.e., the set of all valid firing sequences on N_1 is a subset of all valid firing sequences on N_2. Downsizing is the dual counterpart of upsizing, i.e., N_2 can "do less" than N_1. For example, Fig. 8(4) shows an upsizing. In this case, the SCOC can be constructed by sticking N_1 and N_2 together over *flow-jumpers* (cf. Fig. 8(4)). Flow-jumpers are transitions, which map each marking of N_1 to a marking of N_2 [8]. This way of constructing the SCOC in conjunction with upsizing operations is correct regarding Criterion 5. In the other case—downsizing—the SCOC is constructed by merging N_1 and N_2 over one output place, i.e., instances with tokens in N_1 are further executed according to the old net.

3.3.2. Using complete history information

WIDE Graphs. A widely-used correctness property is the *compliance criterion* introduced by [5]. Intuitively, change Δ of schema S can be correctly propagated to WF instance I on S iff the execution of I, taken place so far, can be "simulated" on the modified schema S' as well. Since WIDE works with a history-based execution model, compliance is based on *trying to replay the execution history* Π_I^S of instance I on the changed schema S'. Formally

Correctness Criterion 6 (*Restrictive compliance criterion*). Let S be a WF schema and I be a WF instance on S with execution history Π_I^S. Let further S be transformed into another schema S' by change operation Δ. Then I is compliant with S' if Π_I^S can be replayed on $S = S + \Delta$ as well, i.e., all events stored in Π_I^S could also have been logged by an instance on S' in the same order as set out by Π_I^S.

Criterion 6 forbids changes of the past (cf. Fig. 9(1)). However, it is too restrictive in conjunction with loops, i.e., it is not loop tolerant as can be seen from Fig. 9(2). Obviously, Π_{I2}^S cannot be produced on S'. Therefore, I_2 is excluded from migration to S' though there would be no problems when proceeding execution of I_2 based on S'. Since [5] gives no information about how to check Criterion 6, we assume that compliance is ensured by trying to replay the whole execution history on the changed schema. Thus, we get the necessary marking adaptations automatically when checking compliance without additional effort. However, doing so causes an overhead due to the possibly extensive volume of history data which is normally not kept in main memory [18].

Similarly, several other approaches [26,27] exist which propose correctness criteria based on instance execution information.

3.3.3. Using a consolidated view on the execution history

TRAMs [18] uses Criterion 6 as well. However, replaying each history entry of an instance on the changed schema is considered as too inefficient, especially when a large number of instances has to be migrated. Therefore, TRAMs provides migration conditions based on which compliance of a WF instance with the changed schema can be checked more efficiently (cf. Table 1).

Due to the declarative control flow definition and the absence of explicit control edges, the insertion of activities is a complex change; i.e., first a new activity node A is inserted and then it is embedded into the control flow by setting the start conditions of A ("incoming edges") and the intended successors ("outgoing edges"). Fig. 10(1) depicts the aggregated migration conditions for

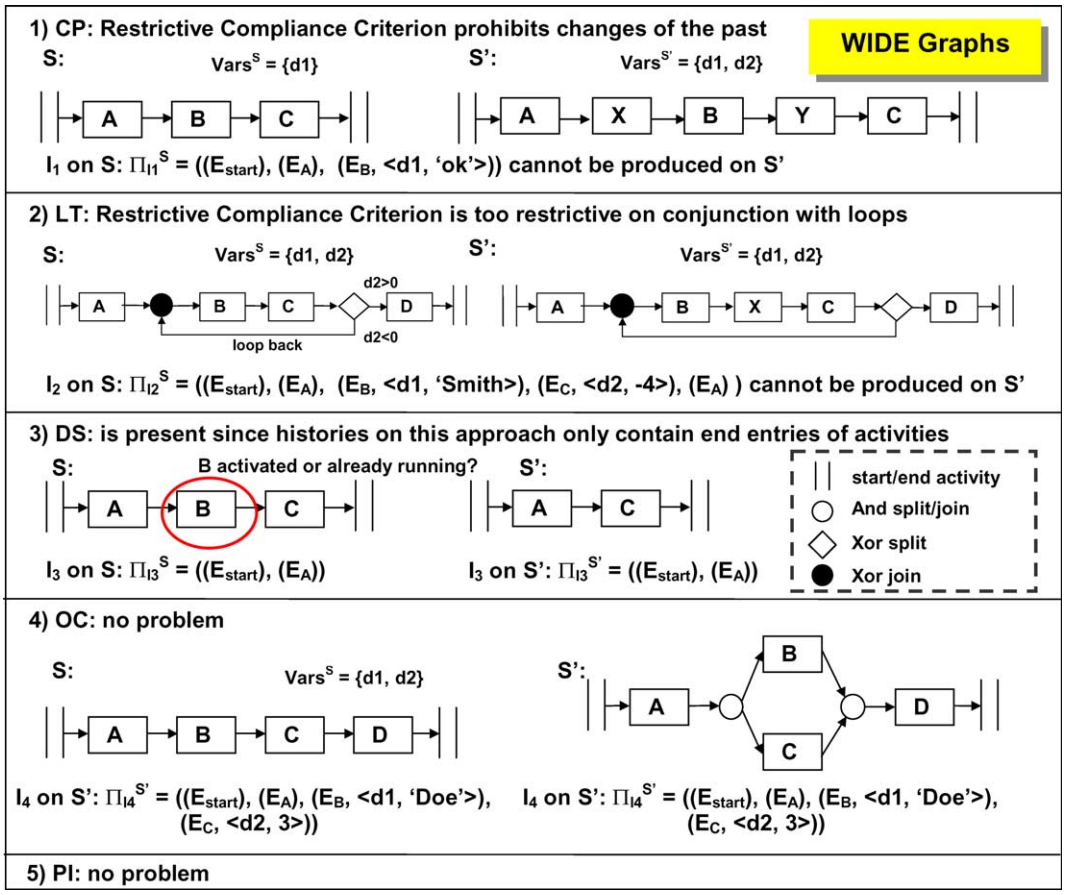

Fig. 9. Checking compliance by replaying the complete execution history [5].

the insertion of two activities and a data dependency between them. Fig. 10(4) shows the parallelization of activities. To our best knowledge TRAMs Graphs are acyclic and consequently problem LT cannot be decided on.

3.3.4. Using model-inherent markings

WSM-Nets. ADEPT [21,24] is based on a correctness criterion which works in conjunction with loop-related and data flow changes as well. It takes up Criterion 6 but modifies it to be loop-tolerant. Here, the key to solution is to differentiate between previous and current/future loop iterations; i.e., considerations are restricted to the relevant parts of the execution history (cf. Definition 1).

Definition 1 (*Reduced Execution History* $\Pi^S_{I\,red}$). Let I be a WF instance with execution history Π^S_I. The reduced execution history $\Pi^S_{I\,red}$ is obtained as follows: In the absence of loops $\Pi^S_{I\,red}$ is identical to Π^S_I. Otherwise, it is derived from Π^S_I by discarding all history entries related to other loop iterations than the last one (completed loop) or the actual iteration (running loop). (Note that $\Pi^S_{I\,red}$ can be easily produced at the presence of nested loops as well.)

Table 1
Examples of migration conditions in TRAMs (cf. [18])

Change Δ	Migration condition for instance I
Insertion of activity A	None
Modifying start condition sc_A of activity A	$(S_A \notin \Pi_I^S) \vee (S_A \in \Pi_I^S \wedge sc_A \text{ holds})$
Deletion of activity A	$S_A \notin \Pi_I^S$
Insertion of read (write) data edge for Activity A	$S_A \notin \Pi_I^S$ $(E_A \notin \Pi_I^S)$

Underline: Π_I^S denotes execution history of I on S; S_A/E_A: start/end event of activity A

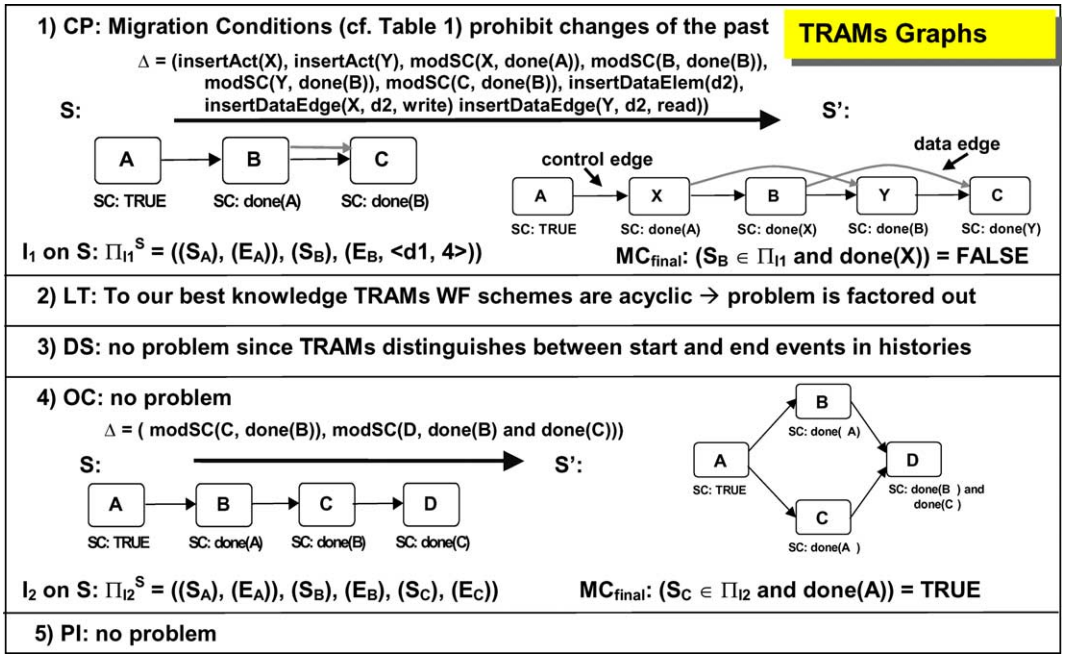

Fig. 10. Migration conditions of TRAMs.

As an example take Fig. 11(2). It shows the reduced execution history of instance I_2 on S. Taking Definition 1 an instance I on S is compliant with a changed schema S' iff the reduced execution history of I can be produced on S' as well.

Correctness Criterion 7 (*Comprehensive compliance criterion*). Let I be a WF instance on WF schema S with execution history Π_I^S and reduced execution history $\Pi_{I\,red}^S$ respectively. Assume further that change Δ transforms S into the correct WF schema S'. Then

- I is compliant with S' iff $\Pi_{I\,red}^S$ can be replayed on S' as well.
- In case of compliance the resulting marking of I on S' is correct.

The challenging question is how to check Criterion 7 without accessing voluminous history data. To guarantee compliance, for each kind of change ADEPT uses quickly checkable marking

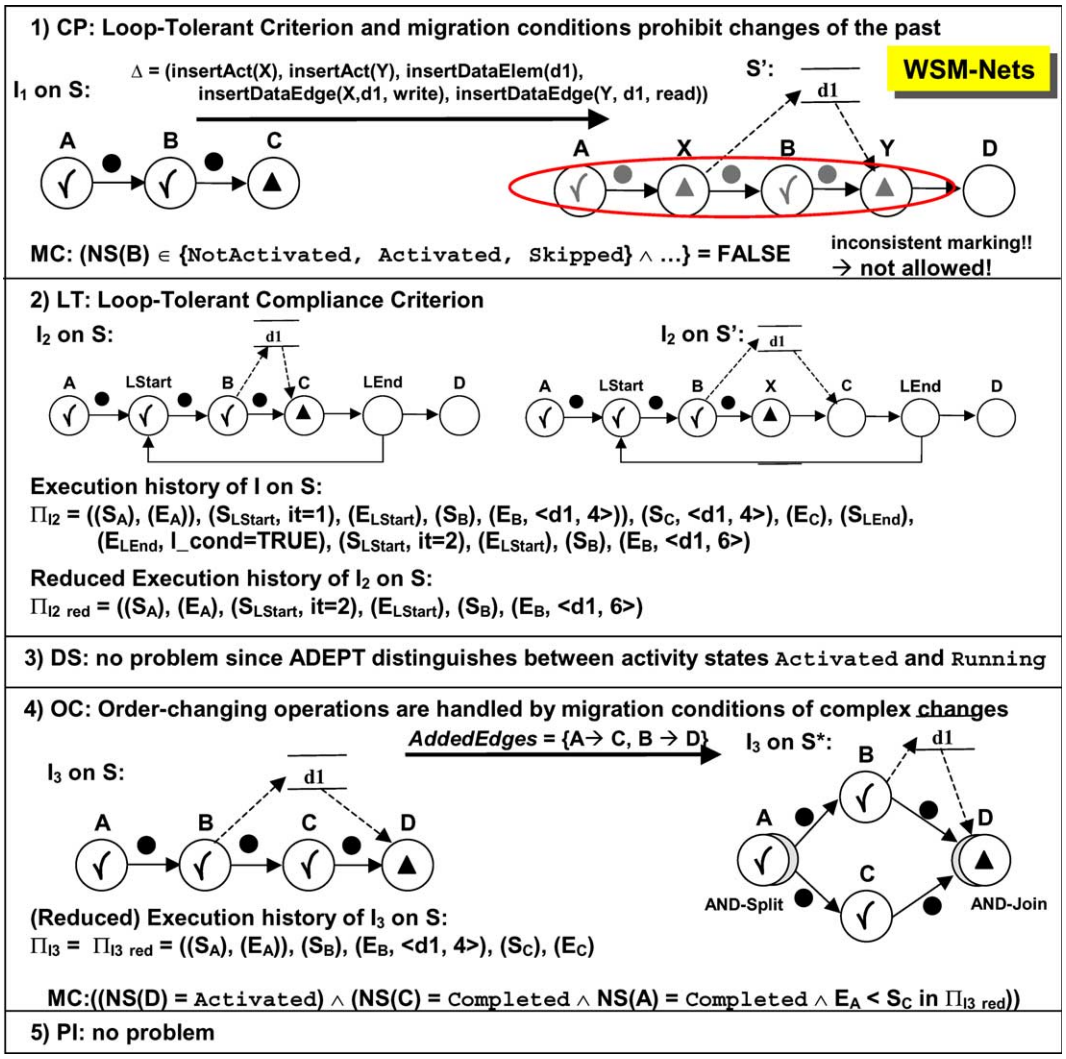

Fig. 11. Analyzing the ADEPT/WSM-Net approach [22–24].

conditions on WSM-Nets. Table 2 exemplarily summarizes conditions for the change operations applied in Fig. 11.

To efficiently adapt markings of compliant instances when migrating them to the changed schema, ADEPT restricts the necessary re-evaluations to those nodes and edges which constitute the *context* of the change region [22]. For each change operation initial node and edge sets to be evaluated are determined. Depending on the result of their marking evaluation the inspection of additional nodes and edges may become necessary. In this context, ADEPT benefits from well-defined marking rules as well as the way node markings are represented in WSM-Nets (preserving markings of passed regions). As an example take change Δ in Fig. 11(2). In the course of the

Table 2
Examples of migration conditions for WSM-Nets (cf. [24])

Change operation Δ	Migration condition for instance I (I compliant with $S' \Longleftrightarrow$)
Control flow change operations	
Δ inserts an activity n_{insert} with associated control and sync edges	$\forall n \in \{x \in N \mid n_{insert} \rightarrow x \in (CtrlE' \cup SyncE')\}$: $NS(n) \in \{\texttt{NotActivated}, \texttt{Activated}, \texttt{Skipped}\} \vee$ n_{insert} is inserted into an already skipped branch of an XOR-branching
Data flow change operations	
Δ inserts a data element d	None
Δ deletes a data element d	No read or write access on d by an activity with state `Running` or `Completed`
Δ inserts/deletes a read edge $d \rightarrow n$	$NS(n) \in \{\texttt{NotActivated}, \texttt{Activated}, \texttt{Skipped}\}$
Δ inserts/deletes a write edge $n \rightarrow d$	$NS(n) \neq \texttt{Completed}$
Complex change operations	
Δ inserts/deletes a set of control or sync edges *AddedEdges*/*DeletedEdges*	$\forall e = n_{src} \rightarrow n_{dest} \in AddedEdges : NS(n_{dest}) \in \{\texttt{NotActivated}, \texttt{Activated}, \texttt{Skipped}\} \vee (NS(n_{dest}) \in \{\texttt{Running}, \texttt{Completed}\} \wedge NS(n_{src}) = \texttt{Completed} \wedge ((\exists e_i = E_{n_{src}}, e_j = S_{n_{dest}}) \in \Pi^S_{I\,red} \wedge i < j))$

following adaptation, `X` has to be marked as `Activated` and marking of `C` is re-evaluated to `NotActivated`. Both, the migration conditions (cf. Table 2) and the marking adaptation algorithm have been implemented in a powerful prototype.

3.4. Coping with non-compliant instances

Breeze [25,26] introduces a *3-phase modification process* for WF schema evolution. The first phase refers to change definition. In the second one, it is tried to bring the affected instances into conformity with the changed schema. In doing so, instances are grouped with respect to their *compliance* with the changed schema (cf. Criterion 6). For non-compliant instances the *compliance graph* is constructed which serves to migrate these instances to the changed schema as well. This graph consists of compensation activities based on which non-compliant instances are partially rolled back into a compliant state. The third phase handles instance migration. To our best knowledge, a detailed discussion about how compliance of instances can be checked is missing.

An alternative approach supporting *delayed migrations* of non-compliant instances is offered by *Flow Nets* [9]. As an example consider Fig. 8(2). Even if instance I on S is not compliant with S' within the actual iteration of a loop, a delayed migration of I to the new change region is possible when another loop iteration takes place. ADEPT has adopted this concept and suggests keeping such (temporarily) non-compliant instances *pending to migrate* [24].

Finally, in *MOKASSIN* [13], application programmers as well as users are burdened with the job to deal with non-compliant WF instances.

4. Exterminating dynamic change problems—a comparison

As can be seen from Table 3 all presented approaches are based on formal correctness criteria. Obviously, there is a trade-off between complexity of the used WF meta model and the flexibility offered by the system during runtime. The more powerful the meta model is the more complex dynamic WF changes are to handle. Agostini and De Michelis [1] have realized this in a very early stage and therefore vote to keep the meta model as simple as possible in order to achieve a maximum of flexibility. For this reason, for example, loops cannot be modeled in MILANO, but must be handled dynamically (via backward jumps) if need be. Furthermore, MILANO limits adaptability to control flow changes, while data flow is managed at the level of single activities. Obviously, this simplifies the users's view on the process. However, in general, control flow changes cannot be treated in a isolated manner and independently from data flow and other workflow aspects.

Furthermore, Table 3 gives a comparison of the different approaches regarding compliance checking and marking adaptations. In [28], an elegant way for checking compliance as well as for automatically adapting instance markings is presented. $WASA_2$ [34] does not explicitly provide compliance checks. We assume that a valid mapping (cf. Definition 2) is determined by comparing nodes, control flow and data flow edges of the purged instance graph for each instance. Though both, replaying whole execution history (WIDE) and checking migration conditions (TRAMs, ADEPT) can be done with the same complexity, generally, there is a giant different in real effort. Reason is that one has to cope with probably extensive data [18] usually not kept in primary storage. However, replaying the complete history information on the changed schema [5] we get the necessary instance markings free to the door whereas in ADEPT, for example, a marking adaptation algorithm has to be applied.

Table 4 compares the different approaches from Fig. 1 with respect to their ability to solve the *dynamic change problems* (cf. Fig. 3).

(1) *Changing the Past* (*CP*). From Fig. 4 it can be seen that all Petri-Net based approaches with True-Semantics (WF Nets, MILANO Nets, and Flow Nets) allow changes of already passed regions of an instance. As mentioned in Section 3 doing so may cause two problems—incomplete input data when invoking activities and inconsistent instance execution. Such problems have been

Table 3
Comparison of meta models, correctness criteria and marking adaptation

	Expressiveness			Completeness of changes	Formal criteria	Compliance checks	Marking adaptations
	General	Loops	DF				
WF Nets	+	+	n.a.	−	+	++	+
$WASA_2$	+	n.a.	+	+	+	n.a.	n.a.
MILANO	∼	n.a.	n.a.	−	+	n.a.	n.a.
Flow Nets	+	+	+	+	+	n.a.	+
WIDE	+	+	+	+	+	+	+
Breeze	0	n.a.	+	+	0	n.a.	n.a.
TRAMs	+	n.a.	+	+	0	+	n.a.
ADEPT	0	+	+	+	+	+	+

DF: data flow; n.a.: "not addressed"; ∼: "simplicity issues".

Table 4
Comparison by means of five typical change problems (cf. Fig. 3)

	(1) CP	(2) LT	(3) DS	(4) OC	(5) PI
WF Nets	Possibly critical	+	Possibly critical	+	+
$WASA_2$	Prevented	–	Prevented	0	+
MILANO	Possibly critical	–	Possibly critical	+	–
Flow Nets	Possibly critical	?	Possibly critical	+	+
WIDE	Prevented	–	Possibly critical	+	+
Breeze	Prevented	–	?	+	+
TRAMs	Prevented	–	Prevented	+	+
ADEPT	Prevented	+	Prevented	+	+

partially factored out in the presented approaches since data flow is not considered. As an example take instance I_1 on S' in Fig. 5(1). Obviously, the newly inserted activity `X` will never be executed, i.e., the execution state of I_1 is not clearly defined. Assume that WF Nets do not exclude data flow and therefore activities `X` and `Y` can be inserted with the data dependency between them (see Fig. 5(1)). This change would be considered as insertion of a parallel branch (*projection inheritance*) and a token be added to place `data`, but with unclear data semantics. Interestingly, this problem is excluded by Flow Nets since common changes of the past and the future are forbidden (cf. Fig. 8(1), ii). Therefore, only the problem of inconsistent instance execution states remains.

(2) *Loop Tolerance* (*LT*). Many of the presented approaches use acyclic WF models whereby problem LT is factored out. The exclusion of loops, however, is out of touch with practical requirements. As discussed in Section 3, it depends on the exact definition of the pre-change firing sequence (cf. Criterion 5) whether Flow Nets are loop-tolerant or not. Anyway, Criterion 6 is too restrictive in conjunction with loops. Therefore ADEPT uses Criterion 7, which is loop-tolerant.

(3) *Dangling States* (*DS*). This problem of being unable to distinguish between activated and running activities is mainly present in Petri-Net based approaches. Transitions usually represent real-world tasks and consume a certain piece of time. If one of them is deleted the challenging question is how to handle in-progress work associated with this transition. In contrast, approaches which explicitly differ between activity states `Activated` and `Running` like $WASA_2$, TRAMs and ADEPT take care of this and ensure that running activities are not disturbed by dynamic changes.

(4) *Order-Changing* (*OC*). This problem appears in conjunction with correctness criteria where certain WF instances are needlessly excluded from migrating to the changed schema. As shown in Section 3.2.2, strict graph equivalence is too restrictive in certain cases.

(5) *Parallel-Insertion* (*PI*). This problem refers to the necessary marking adaptations when inserting a parallel branch such that no deadlocks occur. The only Petri-Net based approach which presents concrete adaptation rules in this context is offered by WF Nets. The suggested strategies ensure a correct control flow in the sequel. However, with respect to data flow, semantics of the newly inserted tokens remains unclear. In ADEPT, markings are automatically adapted when migrating compliant instances to the changed schema. The respective algorithms [22] are based on the marking and execution rules of WSM Nets with their True/False-Semantics.

5. Change scenarios and their realization in existing approaches

In the previous sections emphasis has been put on fundamental correctness issues related to dynamic WF changes. So far it has been circumstantial whether a single WF instance or a collection of instances is subject to change. In this section we have a closer look at different change scenarios and related requirements. We provide a short categorization of adaptive research WF engines which includes Chautauqua [10], $WASA_2$ [34], Breeze [26], and ADEPT [21] whose basics have been already described in Sections 2–4. In addition, we consider the respective approaches followed by AgentWork [20], EPOS [19], and DYNAMITE [12] as well as the flexibility offered by commercial tools.

5.1. Changes of single WF instances

Adaptations of single WF instances become necessary when exceptional situations occur or the structure of a WF dynamically evolves. Both scenarios can be found, for example, in hospital and engineering environments [12,20]. Besides state-related correctness properties instance-specific changes pose several challenging issues. In particular, change *predictability* influences the way how WF instances are adapted during runtime. Regarding evolving workflows, for example, necessary changes and their scope are often known at buildtime [12,19,20]. Consequently, respective adaptations can be *pre-planned* and *automated*. In contrast *ad-hoc changes* have to be applied as response to unforeseen exceptions [21]. Usually, *user interaction* becomes necessary in order to define the respective runtime change. Of course, we cannot always see WF instance changes in terms of black and white, but the distinction between pre-planned and ad-hoc change contributes to classify existing approaches.

5.1.1. Approaches supporting ad-hoc instance changes

In Breeze and $WASA_2$, instance changes can be defined by the use of a graphical WF editor. Using a WF editor for change definition, however, is only conceivable for *expert users*. If changes shall be definable by *end users* as well, application-tailored user interfaces must be offered to them. Obviously, this requires comprehensive programming interfaces. Only few approaches provide such interfaces [21,34]. ADEPT, for example, offers a change API which enables change definition on WSM Nets at a high semantic level, e.g., to jump forward in the flow or to insert a new step between two sets of activities [21]. Very important in this context is to ensure that none of the guarantees which have been achieved by formal checks at buildtime are violated due to the ad hoc change. Note that this does not only require compliance checks and marking adaptations, but also checks with impact to correctness properties of the WF schema itself (e.g. regarding data flow). Therefore ADEPT uses well-defined correctness properties for WF models, formal pre- and postconditions for change operations, and advanced change protocols [21].

5.1.2. Approaches supporting pre-planned instance changes

Support for automatic WF changes is provided by AgentWork [20], DYNAMITE [12], and EPOS [19], but may be realizable on top of adaptive WfMS like $WASA_2$, ADEPT, or InConcert as well. The overall aim is to reduce error-prone and costly manual WF adaptations. In order to realize automatic WF adaptations, firstly, the WfMS must be able to detect logical failures in

which WF instance changes become necessary. Second, it must determine necessary adaptations, identify the instances to be adapted, correctly introduce the change to them, and notify respective users. This poses many additional issues ranging from the consistent specification of preplanned changes at buildtime up to their concrete realization during runtime. Existing approaches supporting automatic WF instance changes can be classified according to different criteria. The most important one concerns the basic method used for automatic failure detection and for change realization. We distinguish between rule-based, process-driven, and goal-based approaches.

Rule-based approaches use ECA (Event/Condition/Action) models to automatically detect logical failures and to determine necessary WF changes. However, most of them limit adaptations to currently executed activities [4,6]. In contrast, AgentWork [20] enables automatic adaptations of the yet unexecuted regions of running WF instances as well. Basic to this is a temporal ECA rule model which allows to specify adaptations at an abstract level and independently of concrete WF models. When an ECA rule fires, temporal estimates are used to determine which parts of the running WF instance are affected by the detected exception. Respective WF regions are either adapted immediately (predictive change) or—if this is not possible—at the time they are entered (reactive change).

Goal-based approaches formalize process goals (e.g., process outputs). In ACT [3], necessary instance adaptations (e.g., substituting the failed activity by an alternative one) are automatically performed if an activity failure leads to goal violation. EPOS [19], in addition, automatically adapts WF instances when process goals themselves change. Both approaches apply planning techniques to automatically "repair" workflows in such cases. However, current planning methods do not provide complete solutions since important aspects (e.g., treatment of loops) are not considered.

Process-driven approaches restrict the possible variants of WF schemes as well as WF changes in advance. DYNAMITE, for example, uses graph grammars and graph reduction rules for this [12]. Automatic adaptations are performed depending on the outcomes of previous activity executions. Interestingly, process-driven as well as goal-based approaches have been primarily applied in the field of engineering workflows. Both DYNAMITE and EPOS provide build-in functions to support dynamically evolving WF instances.

Instance-specific changes pose several other challenges. For example, one must decide on the *duration* of an instance change. Concerning loop-related adaptations, ADEPT differentiates between loop-permanent and loop-temporary changes [21]. The latter are only valid for the current loop iteration. The handling of such temporary changes is not trivial since permanent changes must not depend on them in order to avoid potential errors. AgentWork [20] even follows a more advanced approach by allowing rule designers to specify temporal constraints indicating how long WF adaptations shall be valid.

5.1.3. Ad-hoc changes in commercial tools

Production WFMS like WebSphere MQ Workflow and Staffware [14] provide powerful process support functions but tend to be very inflexible. Particularly, ad-hoc changes of running WF instances are not supported. Unlike these WFMS, engines such as TIBCO InConcert, SER Workflow, and FileNet Ensemble allow on-the-fly adaptations of in-progress instances. For example, users may dynamically insert or delete activities for a given instance in such a way that

the past of this instance cannot be changed. [2] Though ad-hoc WFMS provide high flexibility, they have failed to adequately support end users. Particularly, they do not support them in defining changes and in dealing with potential side-effects resulting from them (e.g., missing input data of an activity due to the deletion of a preceding step). Since one cannot expect from the end user to cope with such problems, this increases the number of errors and therefore limits the practical usability of respective WFMS.

Case handling systems like FLOWer (Pallas Athena) [30] try to address flexibility issues from another viewpoint. Unlike traditional WFMS, case handling provides a higher operational flexibility and aims at avoiding dynamic changes. More precisely, users are allowed to inspect, add or modify data elements before the activities, which normally produce them, are started. Consequently, the decision about which activities can be executed next is based on the available data rather than on information about the activities executed so far. Since FLOWer allows to distinguish between optional and mandatory data elements, a broad spectrum of processes can be covered with this data-driven approach. FLOWer also enables the definition of causal dependencies between activities. The question remains, whether this mixed view on processes (process-driven, data-driven) contributes to completely avoid dynamic changes.

5.2. Workflow type changes and change propagation

WF schema changes at the type level may become necessary, for example, to adapt business processes to a new law or to realize process optimizations. In any case we are confronted with the problem of how to migrate a potentially large number of WF instances $I_1, \ldots, I_n$ running on the old schema S to the new schema S'. Basically, things seem to be the same as for dynamic changes of single WF instances. However, in addition, we are confronted with the problem that the WF type change may have to be propagated to WF instances whose current execution schema $S_I := S + \Delta$ does not completely correspond to S (due to a previous instance change Δ). To exclude such instances from migrating to the new schema S', however, is out of touch with reality, particularly in case of long-running flows. Interestingly, none of the WF engines supporting WF type changes and change propagation has dealt with this problem so far. In WASA_2, for example, individually modified instances cannot be further adapted to later type change. Chautauqua [10] even does not support changes of single instances at all, since instances of a particular type are always connected to the same Flow Net.

Usually, commercial WfMS do not allow change propagation to in-progress instances when a WF schema is modified at the type level. Instead, simple versioning concepts are used to ensure that already running instances can be finished according to the old schema. One exception is offered by Staffware [14]. However, there are several critical aspects arising in this context. For example, running activities can be deleted without any user information. If the deleted activity is finished the returned results disappear to the nirvana. Furthermore, Staffware suffers from the CP problem which may lead to missing input data and activity program crashes at runtime. Finally, Staffware is by far too restrictive (e.g., insertions before activated tasks are forbidden).

[2] In order to avoid undesired side-effects on other cases, for each instance a private schema is kept.

6. Summary

In this paper we have systematically compared approaches supporting adaptive workflows. We have elaborated their strengths and weaknesses along typical dynamic change problems. In doing so, main emphasis has been put on model properties and correctness criteria since they provide the basis for any adaptive WfMS. Furthermore we have sketched different change scenarios and categorized existing approaches supporting them. Though there has been substantial progress in the field of adaptive WfMS during the last years, there is still enough room for further research. Particularly, usability and implementation issues have not been addressed in sufficient detail. Furthermore, many non-trivial interdependencies among different kind of changes exist which must be carefully understood before we come to a complete solution. For example, current adaptive WfMS do not allow propagating WF type changes to individually modified WF instances. As discussed, however, this is very important for the adequate support of long-running workflows [22].

References

[1] A. Agostini, G. DeMichelis, Improving flexibility of workflow management systems. In: BPM'00, LNCS, vol. 1806, 2000, pp. 218–234.
[2] A. Agostini, G. DeMichelis, A light workflow management system using simple process models, Int. J. Collab. Comp. [16] 335–363.
[3] C. Beckstein, J. Klausner, A planning framework for workflow management, in Proceedings of Workshop Intelligent Workflow and Process Management, Stockholm, 1999.
[4] F. Casati, S. Ceri, S. Paraboschi, G. Pozzi, Specification and implementation of exceptions in workflow management systems, ACM TODS 24 (3) (1999) 405–451.
[5] F. Casati, S. Ceri, B. Pernici, G. Pozzi, Workflow evolution, Data and Knowledge Engineering 24 (3) (1998) 211–238.
[6] D. Chiu, Q. Li, K. Karlapalem, Web interface-driven cooperative exception handling in ADOME, Informations Systems 26 (2) (2001) 93–120.
[7] D. Edmond, A.H.M. ter Hofstede, A reflective infrastructure for workflow adaptability, Data and Knowledge Engineering 34 (3) (2000) 271–304.
[8] C. Ellis, K. Keddara, A workflow change is a workflow, in: BPM '00, LNCS, vol. 1806, 2000, pp. 516–534.
[9] C.A. Ellis, K. Keddara, G. Rozenberg, Dynamic change within workflow systems, in: Proceedings of International ACM Conference COOCS '95, Milpitas, CA, August 1995, pp. 10–21.
[10] C.A. Ellis, C. Maltzahn, The Chautauqua workflow system, in: Proceedings of the International Conference on System Science, Maui, HI, 1997.
[11] H.J. Genrich, P.S. Thiagarajan, A theory of bipolar synchronization schemes, Theoret. Comput. Sci. 30 (3) (1984) 241–318.
[12] P. Heimann, G. Joeris, C. Krapp, B. Westfechtel, DYNAMITE: dynamic task nets for software process management, in: Proceedings of the 18th International Conference Software Engineering (ICSE), Berlin, March 1996, pp. 331–341.
[13] G. Joeris, O. Herzog, Managing evolving workflow specifications, in: Proceedings of International Conference on CoopIS '98, New York City, 1998, pp. 310–321.
[14] B. Kiepuszewski, Expressiveness and suitability of languages for control flow modelling in workflows, Ph.D. Thesis, Queensland University of Technology, Brisbane, 2002. Available from <http://www.tm.tue.nl/it/research/patterns>.
[15] B. Kiepuszewski, A.H.M. ter Hofstede, C.J. Bussler, On structured workflow modelling, in: CAiSE'00, LNCS, vol. 1789, 2000, pp. 431–445.

[16] M. Klein, C. Dellarocas, A. Bernstein (Eds.), Int. J. Collab. Comp. 9 (3–4) (2000) 346–456 (Special issue on adaptive workflow systems).
[17] K. Kochut, J. Arnold, A. Sheth, J. Miller, E. Kraemer, B. Arpinar, J. Cardoso, IntelliGEN: a distributed workflow system for discovering protein–protein interactions, Distrib. Parallel Databases 13 (1) (2003) 43–72.
[18] M. Kradolfer, A. Geppert, Dynamic workflow schema evolution based on workflow type versioning and workflow migration, in: CoopIS '99, Edinburgh, 1999, pp. 104–114.
[19] C. Liu, R. Conradi, Automatic replanning of task networks for process model evolution, in: Proceedings of European Software Engineers Conference, Garmisch-Partenkirchen, Germany, 1993, pp. 434–450.
[20] R. Müller, Event-oriented dynamic adaptation of workflows. Ph.D. Thesis, University of Leipzig, Germany, 2002.
[21] M. Reichert, P. Dadam, $ADEPT_{flex}$-supporting dynamic changes of workflows without losing control, JIIS 10 (2) (1998) 93–129.
[22] M. Reichert, S. Rinderle, P. Dadam, On the common support of workflow type and instance changes under correctness constraints, in: CoopIS '03, LNCS, vol. 2888, Catania, Italy, 2003, pp. 407–425.
[23] S. Rinderle, M. Reichert, P. Dadam, Evaluation of correctness criteria for dynamic workflow changes, in: BPM'03, LNCS, vol. 2678, Eindhoven, The Netherlands, 2003, pp. 41–57.
[24] S. Rinderle, M. Reichert, P. Dadam, Flexible support of team processes by adaptive workflow systems, Distribut. Parallel Databases 2004 (to appear).
[25] S. Sadiq, Handling dynamic schema changes in workflow processes, in: Proceedings of the 11th Australian Database Conference 2000.
[26] S. Sadiq, O. Marjanovic, M. Orlowska, Managing change and time in dynamic workflow processes, IJCIS 9 (1–2) (2000) 93–116.
[27] W.M.P.v.d. Aalst, Exterminating the dynamic change bug: a concrete approach to support worfklow change, Inform. Syst. Frontiers 3 (3) (2001) 297–317.
[28] W.M.P.v.d. Aalst, T. Basten, Inheritance of workflows: an approach to tackling problems related to change, Theoret Comp. Sci. 270 (1–2) (2002) 125–203.
[29] W.M.P.v.d. Aalst, M. Weske, G. Wirtz, Advanced topics in workflow management: Issues, requirements, and solutions, Int. J. Integrat. Design Process Sci. 7 (3) (2003).
[30] W.M.P.v.d. Aalst, P. Berens, Beyond workflow management: product-driven case handling, in: Proceedings of the Conference on Supp. Group Work, New York, 2001, pp. 42–51.
[31] W.M.P.v.d. Aalst, S. Jablonski (Eds.), Flexible workflow technology driving the networked economy, Int. Comp. Syst.: Sci. Eng. 15 (5) (2000).
[32] H.M.W. Verbeek, W.M.P.v.d. Aalst, Woflan 2.0 A petri-net-based workflow diagnosis tool, in: ICATPN '00, LNCS, vol. 1825, pp. 455–464, Aarhus, June 2000.
[33] M. Weske, Workflow management systems: Formal foundation, Conceptual design, implementation aspects, University of Münster, Germany, 2000. Habilitation Thesis.
[34] M. Weske, Formal foundation and conceptual design of dynamic adaptations in a workflow management system, in: HICSS-34, 2001.

Stefanie Rinderle studied Mathematics and Economy at the University of Augsburg, Germany. At present she is a Ph.D. candidate of the Databases and Information Systems Department of the University of Ulm, Germany. Her research interests include change management in adaptive workflow management systems, workflow schema evolution, and business process modeling.

Peter Dadam has been full professor at the University of Ulm and director of the Department Databases and Information Systems since 1990. Before he came to the University he had been director of the research department for Advanced Information Management (AIM) at the IBM Heidelberg Science Center (HDSC). At the HDSC he managed the AIM-P project on advanced database technology and applications. Current reserach areas include distributed, cooperative information systems, workflow management, and database technology and its use in advanced application areas.

Manfred Reichert is assistant professor at the Department Databases and Information Systems at the University of Ulm. He finished his Ph.D. thesis on adaptive workflow systems in May 2000. Current research topics include enterprise-wide workflows, enterprise application integration and workflow, process visualization, and different aspects related to workflow technology.