Dealing with Changes of Time-Aware Processes*

Andreas Lanz and Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Germany
{andreas.lanz,manfred.reichert}@uni-ulm.de

Abstract. The proper handling of temporal process constraints is crucial in many application domains. Contemporary process-aware information systems (PAIS), however, lack a sophisticated support of time-aware processes. As a particular challenge, the execution of time-aware processes needs to be flexible as time can neither be slowed down nor stopped. Hence, it should be possible to dynamically adapt time-aware process instances to cope with unforeseen events. In turn, when applying such dynamic changes, it must be re-ensured that the resulting process instances are *temporally consistent*; i.e., they still can be completed without violating any of their temporal constraints. This paper presents the ATAPIS framework which extends well established process change operations with temporal constraints. In particular, it provides pre- and post-conditions for these operations that guarantee for the temporal consistency of the changed process instances. Furthermore, we analyze the effects a change has on the temporal properties of a process instance. In this context, we provide a means to significantly reduce the complexity when applying multiple change operations. Respective optimizations will be crucial to properly support the temporal perspective in adaptive PAIS.

1 Introduction

Time is a crucial factor regarding the proper support of business processes [10]. Moreover, in many application areas (e.g., patient treatment, automotive engineering), the handling of *temporal constraints* is vital in order to successfully execute and complete processes [3,4,10]. However, contemporary process-aware information systems (PAIS) lack a comprehensive support of such *time-aware processes* [10]. To remedy this drawback, the proper integration of temporal constraints with both the design and run-time components of a PAIS has been identified as a key challenge [3,4,7]. Our ATAPIS framework aims to provide comprehensive support for the specification, execution and monitoring of time-aware processes in adaptive PAIS.

As a prerequisite for robust process execution in PAISs, the executable *process models* must be *sound* [12]. Moreover, in the context of *time-aware process models*, i.e., process models enriched with temporal constraints, the *consistency* of the temporal constraints must be ensured [1,4,7]. Checking consistency of time-aware

* A more complete and formally rigor version of this work is described in a technical report [8]

process models at design time has been extensively studied in literature [1,3,5]. By contrast, only little attention has been paid to the proper run-time support of time-aware processes [7]. During run time, the temporal consistency of process instances needs to be continuously monitored and re-checked to avoid constraint violations. Particularly, note that activity durations and deadlines are specific to the executed process instance and only become known at run time [7].

As a particular challenge, temporal constraints cannot be considered in isolation, but might interact with each other. Hence, complex algorithms are required for checking the temporal consistency of a process model [7,15]. At run time, however, respective calculations should be reduced to a minimum to ensure scalability of the PAIS [7]. Otherwise, no run-time support of time-aware processes will be possible at the presence of a large number of process instances.

As another challenge, time can neither be slowed down nor stopped. Accordingly, time-aware processes need to be *flexible* to cope with unforeseen events or delays during run time [14]. For example, it is common that deadlines are re-scheduled or temporal constraints are dynamically modified in order to successfully complete a process instance being in trouble. Moreover, in certain scenarios the instances of time-aware processes must be structurally changed (e.g., by moving, deleting or inserting activities) to be able to meet a particular deadline. In the context of such *dynamic process changes*, we must re-ensure that the resulting process instances are sound and temporally consistent. While soundness has been extensively studied in literature [13,12], this work shows how temporal consistency of a time-aware process instance can be efficiently ensured in the context of dynamic changes. Furthermore, we analyse the effects, changes have on the temporal constraints of the respective process instance. In particular, we show how the results of this analysis can be utilized to significantly reduce the complexity when applying multiple change operations. For example, the latter becomes crucial in the context of process evolution, where a possibly large set of process instances needs to be migrated on-the-fly to a changed process model [12].

The remainder of the paper is organized as follows: Sect. 2 considers existing proposals relevant for our work. Sect. 3 provides background information on time-aware processes and defines the notion of *temporal consistency*. Sect. 4 first introduces the set of change operations we consider, followed by an in-depth discussion on how these change operations work in the context of time-aware processes. Sect. 5 analyzes the impact a change has on the temporal constraints of a process and proposes useful optimizations. Sect. 6 evaluates the proposed approach. Finally, Sect. 7 concludes with a summary and outlook.

2 Related Work

In literature, there exists considerable work on managing temporal constraints for business processes [1,3,5,7,11]. The focus of these approaches is on design-time issues like the modeling and verification of time-aware processes. By contrast, only few approaches consider run-time issues of time-aware processes [4,7]. In particular, none of the latter considers dynamic changes in this context.

Category I: Durations and Time Lags	
TP1	Time Lags between two Activities
TP2	Durations
TP3	Time Lags between Events

Category II: Restricting Execution Times	
TP4	Fixed Date Elements
TP5	Schedule Restricted Elements
TP6	Time-based Restrictions
TP7	Validity Period

Category III: Variability	
TP8	Time-dependent Variability

Category IV: Recurrent Process Elements	
TP9	Cyclic Elements
TP10	Periodicity

Table 1. Process Time Patterns TP1 – TP10 [10]

Most approaches dealing with the verification of time-aware processes use a specifically tailored time model to check for the temporal consistency of process models. This becomes necessary since the interdependencies between the various temporal constraints of a process model can be quite complex and cannot be suitably captured in the respective process model. A specific conceptual model for temporal constraints is defined in [11]. In turn, [4,5] use an extended version of the *Critical Path Method* known from project planning. *Simple Temporal Networks* (STN) are used as basic formalism in [1], whereas [7] uses *Conditional Simple Temporal Networks with Uncertainty* for checking the *controllability* of process models, i.e., a more restrictive form of temporal consistency. This paper relies on *Conditional Simple Temporal Networks* (CSTN), an extension of STN that allows for the proper handling of exclusive choices [15].

In [10], we presented 10 empirically evidenced time patterns (TP), that represent temporal constraints of time-aware processes (cf. Tab. 1). In particular, time patterns facilitate the comparison of existing approaches based on a universal set of notions with well-defined semantics [9]. Moreover, [9,10] elaborated the need for a proper run-time support of time-aware processes.

Dynamic process changes were extensively studied in the past. Particularly, there exists considerable work on ensuring structural and behavioural soundness in the context of dynamic process changes [13]. A survey of approaches enabling dynamic changes is provided in [12]. To the best of our knowledge, [14] is the only work considering dynamic changes in the context of time-aware processes. As opposed to our work, however, [14] only provides a high level discussion of the different aspects to be considered when changing time-aware process instances, temporal consistency being one of them.

3 Basic Notions

This section provides basic notions. First, it defines a set of elements for modeling time-aware processes. Second, it introduces the notion of temporal consistency.

3.1 Time-aware Processes

For each business process exhibiting temporal constraints, a *time-aware process schema* needs to be defined (cf. Fig. 1). In our work, a process schema corresponds to a *process model*; i.e., a directed graph, that comprises a set of *nodes*—representing *activities* and *control connectors* (e.g., Start-/End-nodes, XORsplits, or ANDjoins)—as well as a set of *control edges* linking these nodes and specifying precedence relations between them. We assume that process models

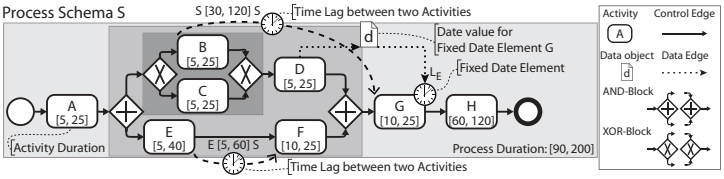


Fig. 1. Core Concepts of a Time-Aware Process Model

are well structured [12], e.g., sequences and branchings are specified in terms of nested single-entry single-exit (SESE) blocks. Fig. 1 depicts an example of a well structured process model with the grey areas indicating respective blocks. Each process model contains a unique start and end node, and may be composed of control flow patterns like sequence, parallel split (ANDsplit), synchronization (ANDjoin), exclusive choice (XORsplit), and simple merge (XORjoin) (cf. Fig. 1).

At run time, *process instances* may be created and executed according to the defined process model. We assume that a process instance is logically represented by a clone of the respective process model augmented with instance-specific information. If a process model contains XOR-blocks, uncertainty is introduced since not all instances perform exactly the same set of activities. The concept of *execution path* allows us to identify which activities and control connectors are actually performed during the execution of a particular process instance.

We base our ATAPIS framework on the time patterns (TP) (cf. Sect. 2). Specifically, we focus on the patterns being most relevant in practice [10]. In detail:

An **activity duration** (TP2) defines the minimum and maximum time span $[d_{min}, d_{max}]$ $(0 \leq d_{min} \leq d_{max})$ allowed for executing a particular activity (or node, in general). We assume that each activity has an assigned duration. Since control connectors are automatically executed, we may assume a fixed duration for them (e.g., $[0, 1]$). In turn, a **process duration** $[d_{min}, d_{max}]$ represents the time span allowed for executing a process instance.

Time lags between two activities (TP1) restrict the time span allowed between the starting and/or ending instants of two arbitrary activities of a process model [10]. In Fig. 1, a time lag is visualized through a dashed edge between the source and target activity. The label of the edge specifies the constraint according to the following template: $\langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle$ $(-\infty \leq t_{min} \leq t_{max} \leq \infty)$; $\langle I_S \rangle, \langle I_T \rangle \in \{S, E\}$ mark the instant (i.e., starting or ending) of the source and target activity the time lag applies to. In turn, $[t_{min}, t_{max}]$ represents the range allowed for the time span between instants $\langle I_S \rangle$ and $\langle I_T \rangle$. Finally, note that a control edge implicitly represents an $E[0, \infty]S$ time lag between the two activities.

Fixed date elements (TP4) allow restricting activity execution in relation to a specific date (e.g., a deadline). Generally, the value of a fixed date element is specific to a process instance. Fig. 1 visualizes a fixed date element through a clock symbol attached to the activity. Thereby, label $\langle D \rangle \in \{E_S, L_S, E_E, L_E\}$ represents the activity's earliest start date (E_S), latest start date (L_S), earliest completion date (E_E), or latest completion date (L_E).

Fig. 1 shows an example of a process model exhibiting temporal constraints. Note that, although some of the symbols used for visualizing the temporal con-

straints resemble BPMN timer events, their semantics is quite different and should not be mixed up.

3.2 Temporal Consistency of Time-Aware Processes

A time-aware process model is executed by performing its activities and control connectors, while obeying a set of temporal constraints. We denote a process model as *temporally consistent* if it is possible to perform all *execution paths* without violating the temporal constraints involved. Temporal consistency of a time-aware process model (and its instances) constitutes a fundamental prerequisite for its robust and error-free execution [1,4]. For any PAIS supporting time-aware processes, therefore, a crucial task is to check temporal consistency of the process model at design time as well as to monitor and re-check corresponding instances during run time. This is particularly challenging since temporal constraints might interact with each other resulting in complex interdependcies (e.g., a future deadline might restrict the duration of some or all preceding activities).

Whether a time-aware process model is temporally consistent can be checked by mapping it to a *conditional simple temporal network* (CSTN)—a problem known from artificial intelligence [6]. In ATAPIS, we use CSTN since it allows us to exploit and reuse *checking algorithms* for a well founded model representing temporal constraints. Finally, CSTN allows capturing the complex interdependencies between constraints, which cannot be captured in process models.

Definition 1 (Conditional Simple Temporal Network). *A Conditional Simple Temporal Network (CSTN) is a 6-tuple* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$*, where:*

- $\mathcal{T}$ *is a set of real-valued variables, called time-points;*
- P *is a finite set of propositional letters (or propositions);*
- $L : \mathcal{T} \rightarrow P^*$ *is a function assigning a label to each time-point in* $\mathcal{T}$*; a label is any (possibly empty) conjunction of (positive or negative) letters from* P*.*[1]
- $\mathcal{C}$ *is a set of labeled simple temporal constraints (constraint in the following); each constraint* $c_{XY} \in \mathcal{C}$ *has the form* $c_{XY} = \langle [x, y]_{XY}, \beta \rangle$*, where* $X, Y \in \mathcal{T}$*,* $-\infty \leq x \leq y \leq \infty$*, and* $\beta \in P^*$ *is a label.*
- $\mathcal{OT} \subseteq \mathcal{T}$ *is a set of observation time-points;*
- $O : P \rightarrow \mathcal{OT}$ *is a bijection that associates a unique observation time-point to each propositional letter from* P*.*

Time-points represent instantaneous events that may be, for example, associated with the start / end of activities. In turn, at *observation time-points* a decision regarding possible execution paths is made. More formally, when executing observation time-point P, the truth-value of the associated proposition (i.e., $O^{-1}(P)$) is determined. A *constraint* $c_{XY} = \langle [x, y]_{XY}, \beta \rangle$ expresses that the time span between time-points X and Y must be at least x and at most y, i.e., $Y - X \in [x, y]$. The *label* attached to each time-point (constraint) indicates possible executions of the CSTN, i.e., a particular time-point (constraint) will be only considered if

[1] In the following we use small Greek letters $\alpha, \beta, \ldots$ to denote arbitrary labels. The empty label is denoted by $\boxdot$.

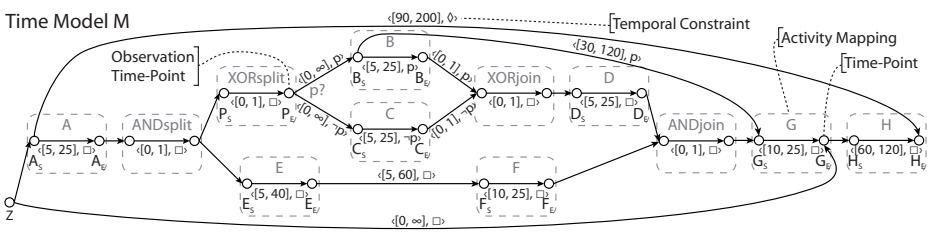


Fig. 2. CSTN Representation of the Process Model from Fig. 1

the corresponding label is satisfiable in the respective instance. Fig. 2 depicts the CSTN corresponding to the process model from Fig. 1.

The solution to a CSTN can be defined as follows [6]:

Definition 2 (Scenario and Solution). *Given a CSTN* $S = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$*, a scenario over set* P *is a function* $s_P : P \rightarrow \{true, false\}$ *that assigns a truth-value to each proposition in* P*.*

A solution for CSTN S *under scenario* s_P *then corresponds to a complete set of assignments to all time-points* $X \in \mathcal{T}$ *with* $s_P(L(X)) = true$*, which satisfies all constraints* $\langle [x, y]_{XY}, \beta \rangle \in \mathcal{C}$ *for which* $s_P(\beta) = true$ *holds.*

We denote the CSTN corresponding to a time-aware process model as its *time model*. The required mapping can roughly be described as follows [7]: First, the control flow of the process model is mapped to a CSTN. Particularly, each control flow element implicitly represents a temporal constraint. Each activity, ANDsplit, ANDjoin, and XORjoin n_i is represented as a pair of time-points N_{iS} and N_{iE}, corresponding to the starting / ending instant of the respective node. In turn, for an XORsplit, the ending instant (i.e., N_{iE}) is represented by an observation time-point. Next, a constraint $\langle [d_{min}, d_{max}]_{N_{iS}N_{iE}}, \boxdot \rangle$ is added between N_{iS} and N_{iE} representing the duration $[d_{min}, d_{max}]$ of the node. Further, for any control edge between nodes n_i and n_j, a constraint $\langle [0, \infty]_{N_{iE}N_{jS}}, \boxdot \rangle$ is added between the time-points representing the ending instant of n_i and starting instant of n_j. If the source of the edge is an XORsplit, in addition, the label of the constraint is augmented by proposition $p = O^{-1}(P)$. The latter represents the decision made at the corresponding observation time-point P, i.e., the label of the constraint $\langle [0, \infty]_{N_{iE}N_{jS}}, \beta \rangle$ belonging to the "`true`"-branch is set to βp and the one of the "`false`"-branch to $\beta \neg p$.[2] Further, the labels of all constraints and time-points corresponding to activities, connectors and control edges in the XOR-block are augmented by either p or $\neg p$ depending on the branch they belong to.

Next, temporal constraints are mapped to the CSTN. A *time lag* $\langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle$ corresponds to a constraint $\langle [t_{min}, t_{max}]_{N_{i\langle I_S \rangle} N_{j\langle I_T \rangle}}, {}_{L(N_{i\langle I_S \rangle}) \wedge L(N_{j\langle I_T \rangle})} \rangle$ between the two time-points representing the respective instants of nodes n_i and n_j. In turn, a *fixed date element* is initially represented as a constraint $\langle [0, \infty]_{ZN_{\langle D \rangle}}, {}_{L(N_{\langle D \rangle})} \rangle$ with Z being a special time-point representing time "0". During run time, value $[0, \infty]$ of the constraint will be updated according to the actual

[2] Note that this can be easily extended to consider more than two branches, but for the sake of simplicity, we only consider two branches in this paper.

fixed date chosen. Finally, process duration $[d_{min}, d_{max}]$ is represented as constraint $\langle [d_{min}, d_{max}]_{N_{0S}N_{kE}}, \boxdot \rangle$ between the time-points representing the starting instant N_{0S} of the first and the ending instant N_{kE} of the last node of the process.

As example consider Fig. 2. Note that the labels of the constraints representing the XOR-block are either set to p or $\neg p$. For the sake of readability, all edges without annotation are assumed to have bounds $\langle [0, \infty], \boxdot \rangle$.

Based on Def. 2, we formally define the notion of *temporal consistency* for time-aware process models.

Definition 3 (Temporal Consistency). *A CSTN $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ is called weakly consistent iff for each scenario s_P at least one solution exists [15].*

A time-aware process model is denoted as temporally consistent iff the corresponding time model (i.e., its CSTN representation) is weakly consistent.

When executing a time-aware process model, temporal consistency of the respective instances needs to be continuously monitored and re-checked. For this purpose, the *minimal network* of a CSTN must be determined.

Definition 4 (Minimal Network). *The minimal network of a CSTN $S = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ is the unique CSTN $M = \langle \mathcal{T}, \mathcal{C}', L, \mathcal{OT}, \mathcal{O}, P \rangle$ having the same set of solutions as S and each value allowed by any constraint $c \in \mathcal{C}'$ being part of at least one solution of S.*

For any CSTN S a minimal network exists iff S is *weakly consistent*. In particular, such a minimal network provides a restricted set of constraints: As long as the value of each time-point is consistent with all constraints referring to it, we can guarantee that the entire CSTN is weakly consistent. Besides *explicit constraints* $c \in \mathcal{C}$ we obtain when mapping the process model to the CSTN, the minimal network contains *implicit constraints* between any pair of time-points that may occur in the same execution path. Note that these implicit constraints represent the effects the explicit constraints have on the overall CSTN (i.e., they represent interdependencies between explicit constraints). The implicit constraints are derived from the explicit ones when determining the minimal network. How to determine the minimal network is described in [15].

When executing a process instance, the minimal network of the time model created at design time is cloned. This *instance time model* is then kept up-to-date with the actual temporal state of the process instance (e.g., deadline, activity start and completion times). Further, it is used to monitor and re-check temporal consistency of the instance [7]. Cloning the time model becomes necessary as the temporal state of each process instance is unique; i.e., no two instances have exactly the same instance time model.

4 Change Operations for Time-aware Processes

Standard change patterns adapting process instances without temporal constraints have been extensively studied in literature [12]. This section discusses how respective change operations may be transferred to time-aware processes. Sect. 4.1

Operation	Informal Description
Control Flow Changes	
$InsertSerial(n_1, n_2, n_{new}, [d_{min}, d_{max}])$	Inserts node n_{new} with duration $[d_{min}, d_{max}]$ between directly succeeding nodes n_1 and n_2.
$InsertPar(n_1, n_2, n_{new}, [d_{min}, d_{max}])$	Inserts node n_{new} with duration $[d_{min}, d_{max}]$ in parallel to the SESE block defined by n_1 and n_2.
$InsertCond(n_1, n_2, n_{new}, [d_{min}, d_{max}], c)$	Inserts node n_{new} with duration $[d_{min}, d_{max}]$ and condition c as well as an XOR block between succeeding nodes n_1 and n_2.
$DeleteActivity(n)$	Deletes activity n.†
Temporal Constraints Changes	
$InsertTimeLag(n_1, n_2, type_{tl}, [t_{min}, t_{max}])$	Inserts a time lag $[t_{min}, t_{max}]$ between nodes n_1 and n_2. Thereby, $type_{tl} \in$ {start-start, start-end, end-start, end-end} describes whether the time lag is inserted between the start of the two activities, the start of n_1 and the end of n_2, the end of n_1 and the start of n_2, or the end of the two activities.
$InsertFDE(n, type_{fde})$	Adds a fixed date element of type $type_{fde} \in \{E_S, L_S, E_E, L_E\}$ to node n.
$DeleteTimeLag(n_1, n_2, type_{tl})$	Deletes the time lag of type $type_{tl}$ between nodes n_1 and n_2.†
$DeleteFDE(n, type_{fde})$	Deletes a fixed date element of type $type_{fde}$ from node n.†

† Delete operations are not considered in this paper, but are discussed in a technical report [8].

Table 2. Basic Change Operations

presents the change operations applicable to time-aware processes. Sect. 4.2 then provides an in-depth discussion of these operations and shows how they can be extended to ensure temporal consistency of a changed process instance.

4.1 Basic Change Operations

When changing a process instance or—more generally—its process model, soundness must be ensured. To achieve this, ATAPIS abstracts from low-level *change primitives* (e.g., adding an edge or node) to higher-level *change operations* with well-defined pre- and post-conditions (e.g., inserting a node serially between two succeeding nodes) [12]. Applied to a sound process model, such a high-level change operation guarantees that the modified process model is structurally and behaviourally sound as well [12]. The upper part of Tab. 2 shows selected change operations required for structurally modifying a process instance. Note that respective operations may be combined to realize more complex change patterns [12] (e.g., move activity). ATAPIS extends the set of structural change operations by change operations that allow modifying the temporal constraints of a process model, e.g., inserting a time lag (see the bottom of Tab. 2). Altogether, the operations allow changing a time-aware process instance, while guaranteeing soundness of the corresponding process model. Due to lack of space, this paper restricts itself to insert operations. A detailed presentation of delete operations is provided in a technical report [8].

4.2 Applying Change Operations to Time-aware Processes

When modifying the model of a time-aware process instance, it must be ensured that the resulting process instance is temporally consistent. This section defines basic criteria ensuring that the application of a change operation does not result in a temporally inconsistent process instance. We further analyze the local impact

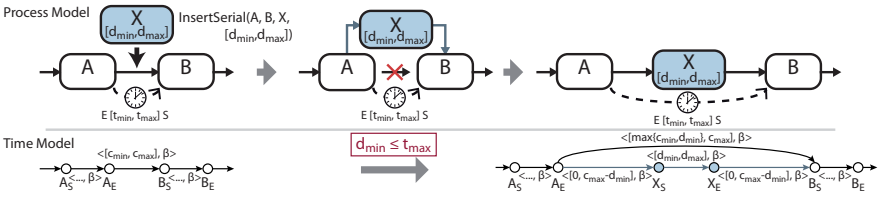


Fig. 3. Change Operation *Insert Serial*

InsertSerial($\mathbf{n_1}$, $\mathbf{n_2}$, $\mathbf{n_{new}}$, $[\mathbf{d_{min}}, \mathbf{d_{max}}]$)*

Pre $succ(n_1) = n_2,\ \forall \langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle \in \mathcal{C} : c_{max} \geq d_{min}$

Init $\gamma = L(N_{1E}) \wedge L(N_{2S})$

Post // Update process model:

...

// Add mapping to instance time model:

$AddTimePoint(N_{newS}, \gamma),\ AddTimePoint(N_{newE}, \gamma),$
$AddConstraint(N_{newS}, N_{newE}, [d_{min}, d_{max}], \gamma),$
$AddConstraint(N_{1E}, N_{newS}, [0, \infty], \gamma),\ AddConstraint(N_{newE}, N_{2S}, [0, \infty], \gamma),$

// Adapt instance time model:

$\forall \langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle \in \mathcal{C} : UpdateConstraint(N_{1E}, N_{newS}, [0, c_{max} - d_{min}], \beta),$
$UpdateConstraint(N_{newE}, N_{2S}, [0, c_{max} - d_{min}], \beta),$
$UpdateConstraint(N_{1E}, N_{2S}, [\max\{c_{min}, d_{min}\}, c_{max})], \beta)$

*The complete version of the algorithm is provided in [8].

Algorithm 1: *InsertSerial*

a particular change operation has on the temporal properties of the respective process model, i.e., its temporal constraints.

When applying a change operation to a process instance, state-specific pre- and post-conditions must be met [12]. Although these are not explicitly considered in this paper, they apply to time-aware processes as well. Furthermore, any time-related, instance-specific data (e.g., activity start and completion times) is maintained in the corresponding *instance time model* (cf. Sect. 3.2), i.e., it is sufficient to only consider the current instance time model of the process instance.

Inserting an Activity Serially. *InsertSerial*$(n_1, n_2, n_{new}, [d_{min}, d_{max}])$ is the first change operation we consider. It allows inserting node n_{new} with duration $[d_{min}, d_{max}]$ between directly succeeding nodes n_1 and n_2 (cf. Fig. 3). Regarding the temporal properties of the resulting process model, the insertion of n_{new} might first and foremost increase the minimum time distance between n_1 and n_2 to d_{min}. By contrast, the maximum distance between the two nodes is not affected by the change as the newly added control connectors do not constrain it. Accordingly, if for the instance time model the minimum duration d_{min} is compliant with any implicit or explicit constraint $\langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle$ between the ending instant of n_1 and the starting instant of n_2 (i.e., $d_{min} \leq c_{max}$), the node insertion will not affect temporal consistency of the process instance.[3] Remember that each value of each constraint in the instance time model is part of at least one solution (cf. Def. 4), i.e., one viable execution of the process model. After adding the node to the process instance, the mapping of this node and the control edges must be added to the instance time model as well. Further, the instance time model must be locally adapted to properly reflect the changes. In particular, the constraint between the ending instant of n_1 and the starting instant of n_2 must be updated to $[\max\{c_{min}, d_{min}\}, c_{max}]$ in order to consider the new minimum

[3] Note that any implicit constraint $\langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle$ is always at least as restrictive as any explicit time lag $E[t_{min}, t_{max}]S$ between n_1 and n_2.

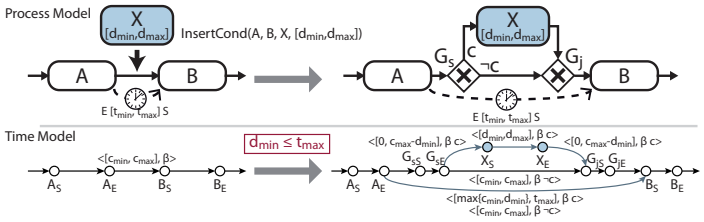


Fig. 4. Change Operation *Insert Conditional*

InsertCond($\mathbf{n_1}$, $\mathbf{n_2}$, $\mathbf{n_{new}}$, [$\mathbf{d_{min}}$, $\mathbf{d_{max}}$], $\mathbf{c}$)*

Pre $succ(n_1) = n_2,\ \forall\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta\rangle \in \mathcal{C} : c_{max} \geq d_{min}$

Init $\gamma = L(N_{1E}) \wedge L(N_{2S})$

Post // Update process model:
...
// Add mapping to instance time model:
$AddTimePoint(G_{sS}, \gamma)$, $AddObservationTimePoint(G_{sE}, c, \gamma)$,
$AddConstraint(G_{sS}, G_{sE}, [0, 1], \gamma)$,
$AddTimePoint(N_{newS}, \gamma)$, $AddTimePoint(N_{newE}, \gamma c)$,
$AddConstraint(N_{newS}, N_{newE}, [d_{min}, d_{max}], \gamma c)$,
...
$AddConstraint(G_{sE}, G_{jS}, [0, \infty], \gamma\neg c)$,
// Adapt instance time model:
...
$\forall\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta\rangle \in \mathcal{C} : UpdateConstraint(N_{1E}, N_{2S}, [c_{min}, c_{max}], \beta\neg c)$,
$AddConstraint(N_{1E}, N_{2S}, [\max\{c_{min}, d_{min}\}, c_{max}], \beta c)$

*The complete version of the algorithm is provided in [8].

Algorithm 2: *InsertCond*

distance between the two nodes (cf. Fig. 3), i.e., certain values permitted by the old constraint might no longer be part of any solution. It further becomes evident that the constraints corresponding to the two newly added control edges must be initialized to $[0, c_{max} - d_{min}]$ (cf. Fig. 3). Algorithm 1 defines the pre- and post-conditions for applying change operation *InsertSerial* to a process instance.

The changes applied to the instance time model need to be propagated to all other constraints in order to remove values no longer contributing to any solution. Note that this must be accomplished before performing any other change or resuming the execution of the process instance. Practically, this means that the minimality of the changed instance time model needs to be restored. This may be achieved by applying the same algorithm as the one initially used for determining the minimal time model (cf. Sect. 3.2).

Inserting an Activity in Parallel. From a temporal point of view, change operation *InsertPar* (cf. Tab. 2) is similar to *InsertSerial*. Node n_{new} (together with ANDsplit and ANDjoin nodes) is inserted "serially" between nodes n_1 and n_2—the temporal effects of the enclosed SESE block are already considered in the implicit constraint between n_1 and n_2. A detailed discussion is provided in [8].

Inserting an Activity Conditionally. Change operation $InsertCond(n_1, n_2, n_{new}, [d_{min}, d_{max}], c)$ inserts node n_{new} conditionally between succeeding nodes n_1 and n_2. This change is accomplished by first inserting XORsplit g_s and XOR-join g_j sequentially between n_1 and n_2 and then n_{new} conditionally between g_s and g_j (cf. Fig. 4). The transition conditions of the control edges linking g_s and its successors are set to c and $\neg c$, respectively. When adding XORsplit g_s and condition $c/\neg c$ to the process model, a set of additional execution paths results; i.e., each execution path of the old process model, which contains n_1 and n_2, can now be mapped to two execution paths: one with $c = false$ (i.e., $\neg c$)

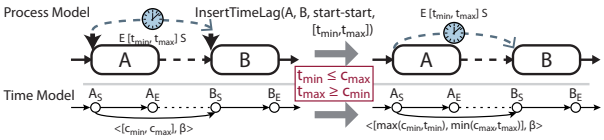


Fig. 5. Change Operation *Insert Time Lag*

InsertTimeLag($\mathbf{n_1}$, $\mathbf{n_2}$, $\mathbf{type_{tl}}$, [$\mathbf{t_{min}}$, $\mathbf{t_{max}}$])

Pre $\langle I_S \rangle = \begin{cases} S & type_{tl} = \text{start-}* \\ E & type_{tl} = \text{end-}* \end{cases}$, $\langle I_T \rangle = \begin{cases} S & type_{tl} = *\text{-start} \\ E & type_{tl} = *\text{-end} \end{cases}$
$(L(N_{1\langle I_S \rangle}) \wedge L(N_{2\langle I_T \rangle}))$ is satisfiable
$\forall \langle [c_{min}, c_{max}]_{N_{1\langle I_S \rangle} N_{2\langle I_T \rangle}}, \beta \rangle \in \mathcal{C} : c_{min} \leq t_{max} \wedge t_{min} \leq c_{max}$

Post // Update process model:
$AddTimeLag(n_1, n_2, \langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle)$
// Add mapping to instance time model:
$AddConstraint(N_{1\langle I_S \rangle}, N_{2\langle I_T \rangle}, [t_{min}, t_{max}], L(N_{1E}) \wedge L(N_{2S}))$
// Adapt instance time model:
$\forall \langle [c_{min}, c_{max}]_{N_{1E} N_{2S}}, \beta \rangle \in \mathcal{C}$:
$UpdateConstraint(N_{1\langle I_S \rangle}, N_{2\langle I_T \rangle}, [\max\{c_{min}, t_{min}\}, \min\{c_{max}, t_{max}\}], \beta)$

Algorithm 3: *InsertTimeLag*

representing the previous execution path and one with $c = true$ representing the new path containing n_{new} between n_1 and n_2. Hence, for any execution path containing n_{new}, *InsertCond* has similar effects as *InsertSerial*. In turn, any execution path not containing n_{new} remains unchanged (except for the added XORsplit and XORjoin, that constitute silent nodes). Altogether, for *InsertCond* similar pre-conditions as for *InsertSerial* hold (cf. Algorithm 1).

In the context of a process instance change, the corresponding instance time model needs to be adapted by adding the mappings of the inserted elements as shown in Fig. 4. Note that this results in a new observation time-point G_{sE} and proposition c to the instance time model (cf. Sect. 3.2). Accordingly, the labels of the temporal constraints representing n_{new} and the two control edges connecting it with g_s and g_j must be set to βc with β being the label of the original constraint between N_{1E} and N_{2S}. In turn, the label of the constraint corresponding to the control edge between g_s and g_j must be set to $\beta \neg c$. Finally, the constraint between the ending instant of n_1 and the starting one of n_2 needs to be updated: The label of the original constraint must be augmented by proposition $\neg c$ resulting in constraint $\langle [c_{min}, c_{max}]_{N_{1E} N_{2S}}, \beta \neg c \rangle$. Further, another constraint $\langle [\max\{c_{min}, d_{min}\}, c_{max}]_{N_{1E} N_{2S}}, \beta c \rangle$ containing proposition c must be added between the two time-points. The latter corresponds to the case n_{new} is executed between the two nodes. Algorithm 2 defines the pre- and post-conditions of *InsertCond*. When applying this operation, again the minimality of the adapted instance time model must be restored. This is required before performing any other change or resuming the execution of the process instance.

Inserting a Time Lag. Operation $InsertTimeLag(n_1, n_2, type_{tl}, [t_{min}, t_{max}])$ allows adding a time lag between activities n_1 and n_2. The instants the time lag refers to are specified by parameter $type_{tl}$. Adding a time lag is only possible if there exists at least one execution path containing both nodes [9]. The instance time model is then adapted by adding a constraint $\langle [t_{min}, t_{max}]_{N_{1\langle I_S \rangle} N_{2\langle I_T \rangle}}, \beta \rangle$ between the time-points representing the respective instants (start vs. end) of the two nodes. Basically, this updates each implicit constraint $\langle [c_{min}, c_{max}]_{N_{1\langle I_S \rangle} N_{2\langle I_T \rangle}}, \beta \rangle$. Note that this is only possible if the resulting constraint $[\max\{c_{min}, t_{min}\},$

$\min\{c_{max}, t_{max}\}]$ in the adapted instance time model still permits at least one value, i.e., it allows for at least one possible solution. Accordingly, in order to apply the operation it must hold $c_{min} \leq t_{max} \wedge t_{min} \leq c_{max}$. Algorithm 3 defines the pre- and post-conditions. After updating the temporal constraints, minimality of the adapted instance time model must be restored.

Inserting a Fixed Date Element. Inserting a fixed date element (i.e., operation *InsertFDE*) is equivalent to adding a time lag between the special time-point Z (indicating time "0") and the respective instant of the node (cf. Sect. 3.2) [8].

5 Analyzing the Effects of Change Operations

When changing a time-aware process instance both the process model and the instance time model must be updated. In this context, the minimality of the instance time model must be restored after each change operation. Only then it can be ensured that another change within the same change transaction may be applied without violating temporal consistency of the process instance. However, calculating the minimal network of a CSTN is expensive regarding computation time, i.e., its complexity is $O(n^3 2^k)$ with n being the number of time-points and k the number of observation time-points in the CSTN. Consequently, there might be significant delays when applying multiple change operations to large time-aware process instances. This becomes even more pressing in the context of process schema evolution [12] when migrating a potentially large set of process instances to a new schema version (i.e., process model). Hence, the maximum effect a particular change has on the instance time model must be estimated. Based on this estimation, it becomes possible to decide whether another change operation may be applied without need to restore minimality of the instance time model first.

When applying the change operations from Sect. 4.2 to the respective instance time model, two types of changes result: adding a temporal constraint or making an existing one more restrictive. Hence, it is sufficient to consider the effects a basic change has on a minimal time model. Regarding changes that make an existing constraint more restrictive, Theorem 1 shows how their maximum effects can be estimated.

Theorem 1 (Restricting a constraint in a minimal network). *Let $M = \langle \mathcal{T}, \mathcal{C}_M, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be a minimal CSTN and $M^* = \langle \mathcal{T}, \mathcal{C}_{M^*}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ the CSTN derived from M by replacing constraint $c_{AB} = \langle [x, y]_{AB}, \beta \rangle \in \mathcal{C}_M$ with the more restrictive constraint $c^*_{AB} = \langle [x + \sigma, y - \rho]_{AB}, \beta \rangle$; $\sigma, \rho \geq 0$; i.e., $C^*_M = C_M \setminus c_{AB} \cup \{c^*_{AB}\}$.*

Then: For the minimal network $N = \langle \mathcal{T}, \mathcal{C}_N, L, \mathcal{OT}, \mathcal{O}, P \rangle$ of M^* it holds: for any constraint $c'_{XY} = \langle [x', y']_{XY}, \gamma \rangle \in \mathcal{C}_N$ the lower bound is increased by at most $\delta = \max\{\sigma, \rho\}$ and the upper bound is decreased by at most δ compared to the original constraint $c_{XY} = \langle [x, y]_{XY}, \gamma \rangle \in \mathcal{C}_M$. Formally:

$$\forall \langle [x, y]_{XY}, \gamma \rangle \in \mathcal{C}_M, \langle [x', y']_{XY}, \gamma \rangle \in \mathcal{C}_N : (x \leq x' \leq x + \delta) \wedge (y \geq y' \geq y - \delta)$$

A proof of Theorem 1 can be found in [8]. Assume that due to a change a constraint $[x, y]_{XY}$ in the time model is restricted to $[x^*, y^*]_{XY} = [x + \rho, y - \sigma]_{XY}$

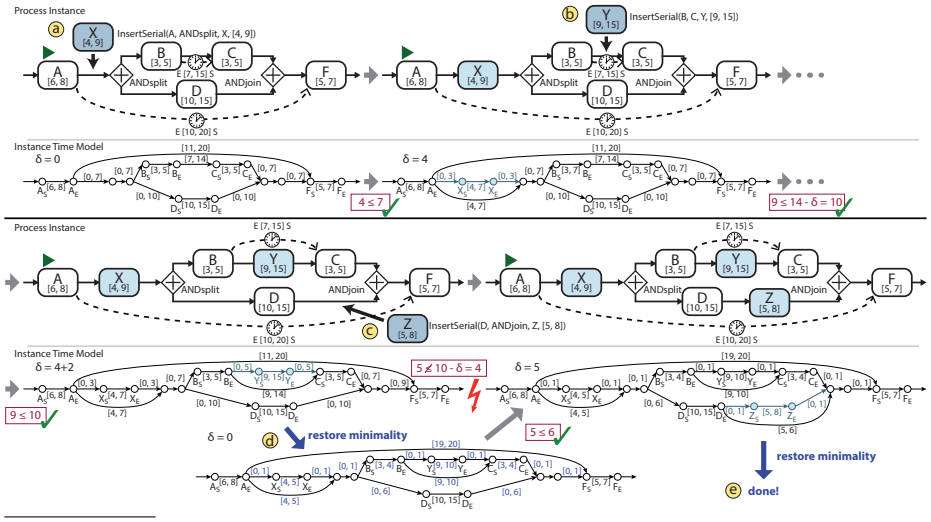
and afterwards minimality of the time model is restored. Theorem 1 now states that any constraint $[u, v]_{UV}$ in the original time model is restricted to at most $[u', v']_{UV} = [u + \delta, v - \delta]_{UV}$ with $\delta = \max\{\rho, \sigma\}$ in the new time model.

Reconsider operation *InsertSerial*. Assume that the instance time model is adapted as described by Algorithm 1. The next step would be to restore minimality of this instance time model. First of all, note that the constraints introduced by the newly added activity and control edges do not affect the other constraints when restoring minimality. By construction, their effects are already incorporated in the constraint between time-points N_{1E} and N_{2S}, which is updated in the context of the operation (cf. Algorithm 1; see [2] for details). The only change having an effect on the resulting instance time model is the one restricting constraint $[c_{min}, c_{max}]$ between N_{1E} and N_{2S} to $[\max\{c_{min}, d_{min}\}, c_{max}]$. Note that if the constraint is not changed (i.e., $d_{min} \leq c_{min}$), the existing constraints of the instance time model also need not be changed. Otherwise, the lower bound of the constraint is increased by $\delta = d_{min} - c_{min}$. Theorem 1 implies that the upper and lower bound of any other constraint in the new instance time model will be restricted by at most δ as well. Thus we are able to approximate the maximum difference between the new instance time model and the original one.

From this we can conclude that when applying another insert operation, it will be sufficient to verify that any precondition referring to a constraint $\langle [x, y]_{XY}, \beta \rangle$ of the instance time model is satisfied for the respective approximated constraint $\langle [x + \delta, y - \delta]_{XY}, \beta \rangle$ as well. In this case, the insert operation may be applied without violating the temporal consistency of the process instance. In particular, and this is a fundamental advantage of ATAPIS, we need not restore minimality of the modified instance time model prior to the application of the operation. By contrast, if the precondition is not met for the approximated constraint, it might still be possible to apply the change without violating temporal consistency. In this case, however, minimality of the modified instance time model must be first restored before deciding whether the change may be applied.

Similar rules apply to all other insert operations. Regarding *InsertCond* (cf. Algorithm 2), in particular, the change relevant to the instance time model is the one restricting the constraint between time-points N_{1E} and N_{2S} to $[\max\{c_{min}, d_{min}\}, c_{max}]$, i.e., the impact on the other constraints is at most $\delta = \max\{0, d_{min} - c_{min}\}$. Finally, for *InsertTimeLag*, the maximum impact corresponds to $\delta = \max\{0, t_{min} - c_{min}, t_{max} - c_{max}\}$ (cf. Algorithm 3).

Based on these observations it becomes possible to apply a sequence of change operations to a process instance within the same transaction without need to restore minimality of the instance time model after each change. If a sequence of change operations $op_1, \ldots, op_n$ with impacts $\delta_1, \ldots, \delta_n$ shall be applied to a process instance, it will be sufficient to consider the aggregated impact of the previously applied operations. Practically speaking, for operation op_i, approximated constraint $[x + \sum_{j=1}^{i-1} \delta_j, y - \sum_{j=1}^{i-1} \delta_j]_{XY}$ needs to be considered to determine whether the operation may be applied. Note that this will significantly reduce complexity when applying multiple change operations. However, the actual savings depend on the strictness of the constraints of the time-aware process



*Note that for sake of compactness only relevant constraints and no labels are shown for the instance time models.

Fig. 6. Applying Multiple Change Operations to a Process Model

model; if the latter is "heavily" constrained, only few change operations can be applied without need to restore minimality of the instance time model. In turn, if the constraints are "weak", multiple change operations may be applied at once, without having to restore minimality of the instance time model between changes.

We illustrate our approach along the example from Fig. 6. It depicts a process instance and corresponding instance time model to which a series of three change operations ⓐ-ⓒ shall be applied. First, activity `X` with duration $[4, 9]$ shall be inserted between `A` and `ANDsplit` (Fig. 6 ⓐ). This is possible without violating the temporal consistency of the process instance since the minimum duration of `X` is lower than the maximum time distance between `A` and `ANDsplit` (i.e., $4 \leq 7$). After performing the change, the value used for approximating the instance time model becomes $\delta = 4-0 = 4$. Next, `Y` shall be inserted between `B` and `C` (Fig. 6 ⓑ). Again this is possible since the minimum duration is lower than the approximated maximum time distance (i.e., $9 \leq 14 - \delta = 10$). Afterwards δ is increased to $\delta = 4 + (9 - 7) = 6$. However, inserting `Z` with duration $[5, 8]$ between `D` and `ANDjoin` (Fig. 6 ⓒ) is then not possible based on the approximated instance time model as the precondition of the respective change operation cannot be met (i.e., $5 \not\leq 10 - \delta = 4$). Hence, minimality of the instance time model must be restored (Fig. 6 ⓓ). Afterwards, inserting `Z` becomes possible as for the new instance time model the precondition of the operation is met. Finally, minimality of the last instance time model must be restored (Fig. 6 ⓔ).

6 Proof of Concept

The presented approach was implemented as a proof-of-concept prototype in our ATAPIS Toolset, which is based on the AristaFlow BPM Suite [12]. This prototype enables users to create time-aware process models and to automatically

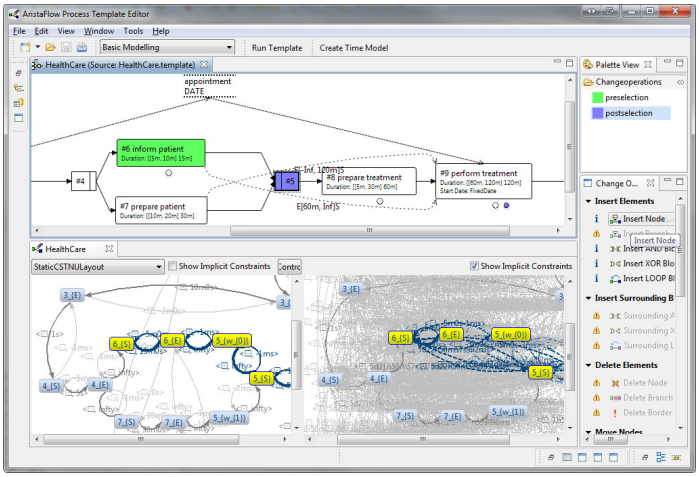


Fig. 7. Screenshot of the Prototype (based on the AristaFlow BPM Suite)

generate respective time models based on CSTN. Further, the presented change operations may be applied to both process models and corresponding instances. Particularly, they are based on AristaFlow's well-founded set of change operations [12]. Overall, the prototype demonstrates the applicability of our approach. The screenshot from Fig. 7 shows the ATAPIS Toolset[4]: at the top, a process model from the healthcare domain comprising several temporal constraints is shown. At the bottom, the automatically generated time model and its minimal network are depicted. Finally, the right side displays the available set of change operations. Whether a particular change operation may be applied is decided by checking both structural and temporal preconditions. When applying an operation to the process model (i.e., schema or instance) all three models are updated simultaneously as described in Sect. 4.1. A first simulation based on our prototype shows a significantly improved performance of our approximation-based approach for applying multiple change operations compared to the "classical approach" [8].

7 Conclusion

Time constitutes a fundamental concept for the operational support of business processes in PAISs. In business, where missed deadlines and violations of temporal constraints might cause significant problems, it is crucial for enterprises to be able to efficiently control and monitor these temporal constraints during run time. Since process execution does not always stick to the plan, enterprises must be further able to flexibly react to deviations in a time-aware process instance without affecting other properties of the instance. This paper considered dynamic changes of time-aware process instances. First, we defined change operations for time-aware processes. Second, we specified pre- and post-conditions for these

[4] A screencast demonstrating the toolset is available at dbis.info/atapis

operations, which ensure that changed process instances remain temporally consistent. Third, we analyzed the effects respective change operations have on the temporal constraints of the process instance. Fourth, we approximated the resulting temporal properties of the entire process instance. In particular, this allows us to significantly reduce the complexity of the required time calculations in the context of subsequent changes. In order to demonstrate the feasibility of the presented approach, a powerful proof-of-concept prototype was implemented.

We are currently investigating the pre- and post-conditions as well as the impact of more complex change patterns (e.g., move). We further will examine how the presented results can be applied to evolve time-aware processes and migrate a large set of process instances to a new process model. Finally, we are integrating advanced time-management capabilities into the AristaFlow BPM Suite to obtain a fully-fledged time- and process-aware information system.

References

1. Bettini, C., Wang, X.S., Jajodia, S.: Temporal reasoning in workflow systems. Distrib Para Dat 11(3), 269–306 (2002)
2. Chen, J., Yang, Y.: Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems. ACM Trans on Soft Eng and Methodol 20(3), 9:1–9:23 (2011)
3. Combi, C., Gozzi, M., Posenato, R., Pozzi, G.: Conceptual modeling of flexible temporal workflows. TAAS 7(2), 19:1–19:29 (2012)
4. Eder, J., Euthimios, P., Pozewaunig, H., Rabinovich, M.: Time management in workflow systems. In: Proc BIS'99. pp. 265–280 (1999)
5. Eder, J., Gruber, W., Panagos, E.: Temporal modeling of workflows with conditional execution paths. In: Proc DEXA'00. pp. 243–253 (2000)
6. Hunsberger, L., Posenato, R., Combi, C.: The dynamic controllability of conditional STNs with uncertainty. In: Proc PlanEx'12 (2012)
7. Lanz, A., Posenato, R., Combi, C., Reichert, M.: Controllability of time-aware processes at run time. In: Proc CoopIS'13. pp. 39–56 (2013)
8. Lanz, A., Reichert, M.: Process change operations for time-aware processes. Tech. Rep. UIB-2014-01, University of Ulm (2014), `dbis.eprints.uni-ulm.de/1027/`
9. Lanz, A., Reichert, M., Weber, B.: A formal semantics of time patterns for process-aware information systems. Tech. Rep. UIB-2013-02, University of Ulm (2013)
10. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Req Eng 19(2), 113–141 (2014)
11. Marjanovic, O., Orlowska, M.E.: On modeling and verification of temporal constraints in production workflows. Knowl and Inf Syst 1(2), 157–192 (1999)
12. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer (2012)
13. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems: A survey. Data & Knowl Eng 50(1), 9–34 (2004)
14. Sadiq, S.W., Marjanovic, O., Orlowska, M.E.: Managing change and time in dynamic workflow processes. Int'l J Coop Inf Syst 9(1-2), 93–116 (2000)
15. Tsamardinos, I., Vidal, T., Pollack, M.: CTP: A new constraint-based formalism for conditional, temporal planning. Constraints 8(4), 365–388 (2003)