SN Computer Science (2020) 1:231
https://doi.org/10.1007/s42979-020-00242-8

ORIGINAL RESEARCH

Conditional Schedules for Processes with Temporal Constraints

Johann Eder[1] · **Marco Franceschetti**[1] · **Josef Lubas**[1]

Received: 4 May 2020 / Accepted: 24 June 2020 / Published online: 14 July 2020

Abstract
Temporal aspects are among the most important quality criteria for executing business processes. It is mandatory to check process definitions at design time for temporal properties to avoid that structural properties of process models cause time failures at run-time. Here, we propose to check process models for conditional controllability and to compute conditional schedules for their execution without time failures. Schedules have to be conditional, since it is a characteristics of business processes, that control flow decisions are based on conditions, which can only be evaluated in the course of process execution at run-time and not before the process starts. We present a procedure for checking the conditional controllability of processes with temporal constraints, which is both sound and complete and effectively and efficiently computes conditional schedules for temporally constrained business processes.

Keywords Process scheduling · Contingent durations · Controllability · Temporal constraints · Process modelling

Introduction

Processes have been successfully introduced for modeling the dynamics in many areas like trade, production, health care, etc. in various forms like workflows [23], extended transactions [29], business processes [13], web-service orchestrations [11], distributed workflows [22], etc. In many of these application areas, temporal aspects are crucial for the correct and admissible execution of processes. This observation led to a substantial body of research to master the plenitude of temporal aspects of process engineering: expressing temporal aspects in process models, formulating different notions of correctness of process models with temporal constraints, checking the temporal correctness of process definitions, computing execution schedules for processes, recognizing and handling temporal exceptions, and supporting process controllers to adhere to temporal constraints at run-time with proactive time management (see [5, 20, 25] for overviews).

What is the right correctness criterion for temporally constrained processes? Satisfiability, or conformance [10, 17] turned out to be not sufficient, respectively, not strong enough, because satisfiability only checks, whether there exists an execution of the process without time failures. However, the execution might depend on factors, which cannot be influenced by the process controller. Hence the process controller cannot guarantee that the execution will be free of time failures. Therefore, the notion of controllability of process definitions [10, 36] gained support. In a nutshell: the concept of controllability regards a process definition as correct, if it is possible to state a schedule such that all temporal constraints are obeyed, if all process steps are executed within the intervals defined in the schedule [9]. Controllability guarantees a correct execution for all foreseeable circumstances. Nevertheless, strict controllability is quite restrictive, such that more relaxed notions were developed, which still hold the guarantee. Conditional controllability (also called history dependent controllability [10, 14]) allows that the execution interval for a step in a conditional schedule depends on the observations of flow decisions, if these flow decisions happen before the activation of this step. Dynamic controllability relaxes this notion even further and requires that there is an execution strategy for a process controller to make decisions based on all observations (flow decisions and observed duration of activities) temporally prior to the enactment of an activity such that no temporal constraint is violated. Similar problems where studied in the area of AI and in the area of constraint satisfaction in form of temporal constraint networks of different flavors as described in [8].

Several algorithms for checking controllability resp. dynamic controllability of process definitions with several

This article is part of the topical collection "Future Data and Security Engineering 2019" guest edited by Tran Khanh Dang.

✉ Johann Eder
johann.eder@aau.at

[1] Department of Informatics-Systems, Alpen-Adria Universität Klagenfurt, Klagenfurt, Austria

sets of temporal constraints have been proposed for different flavors of process models, e.g., [2, 3, 6, 8, 15, 26, 27, 30, 37]. Nevertheless, an algorithm which is computationally feasible, both sound and complete and effectively computes a conditional schedule was still missing. It is the ambition of this paper to contribute to closing this gap. We build on our earlier approaches for checking the correctness of temporal process specifications and the computation of execution plans (schedules) [17–19, 21]. This paper is based on [14] and extends it with several optimizations of the algorithms reducing the average case complexity, increasing practical scalability and improving the performance of computing conditional schedules.

We present our methods with a seemingly rather simple model for defining processes with temporal constraints: we only consider contingent nodes and upper-bound constraints on end-events of activities. This is, however, without loss of generality, as it does not cause any restriction of the expressiveness. We can show, that our lean process model is expressive enough to also express non-contingency of activities, lower-bound constraints, constraints on the start-events of activities, and contingent and non-contingent constraints on links between activities [16].

The particular contributions of this paper are:

- We provide a formalization of conditional controllability.
- We propose a procedure for checking the conditional controllability of temporally constrained process definitions and proof that it is both sound and complete.
- This algorithm also computes a conditional schedule.
- We proof that the problem of computing a conditional schedule for a temporally constrained process has exponential complexity in the worst case.
- We provide optimizations for computing conditional schedules where the complexity corresponds to the intricacy of mutually influencing temporal constraints and structural characteristics.

The rest of this paper is organized as follows: first, we define our process modelling notation. Given that, we introduce schedules and controllability, and in the next section, we present an algorithm for computing schedules. In the following section, we introduce conditional schedules and conditional controllability and the notion of an unfolded graph, and we show the relationship between conditional controllability and the controllability of an unfolded process. After that, we analyse the complexity of conditional schedules. In the following section, we show an improved algorithm for checking conditional controllability with a partially unfolded process graph. Later we present an implementation and report on a series of experiments with the proposed algorithms. In the last sections, we contrast our approach to related work and draw some conclusions.

Process Model with Temporal Constraints

We consider here a minimal but quite expressive process model: we focus on the standard minimum workflow control patterns [35]: acyclic workflow nets composed of nodes and edges. Some of the nodes are XOR-splits and -joins. All other nodes contain implicitly AND-splits and -joins. XOR-splits have exactly 2 outgoing and XOR-joins 1 or 2 incoming nodes. XOR-joins have the semantics of simple merge, i.e., it is not possible that more than predecessors of an XOR-join node will be executed in a single process instance. For all nodes with the exception of XOR-splits: if there are several successor nodes, then the semantics is that of an (implicit) AND-split. For all nodes with the exception of XOR-joins: if there are several predecessors the semantics is that of an AND-join.

This definition of a process graph is a relaxation of the usual workflow net definitions [34] and it is easy to see that all traditional workflow nets can easily be expressed in this formalism. Nevertheless, we introduce the model below, as it makes the transformations we propose in the following sections for checking controllability a lot easier and more comprehensive and does not require to introduce an additional model for the result of unfolding operations.

In addition, we consider the following temporal information and constraints: minimum and maximum duration of nodes, and upper-bound constraints. See Fig.1 as an example for such a process with labels, discussed below. Activities are represented as rectangles with their activity name (e.g., G, their minimum and maximum duration (e.g., 5..8) and their labels (e.g.,$\{p, \neg p\}$. XOR-split nodes are represented as diamonds, XOR-join nodes as shaded diamonds. Upper-bound constraints are represented as dashed arrows from destination to source.

Temporal aspects are represented by durations and time points in form of real numbers, where all durations have to be greater or equal 0. Time points are represented as distance to a time origin (here usually the time point of the start of a process). Activity instances start at a certain time point (time point for the start event of an activity instance) and end at a certain time point (end event); their distance is the duration of an activity instance.

In the following we only consider contingent activities. The duration of contingent activities can only be observed by the process controller, but not influenced. The process controller can only rely on the actual duration being between the minimum and maximum duration specified.

In analogy to [8], we use propositional labels for nodes to indicate through which path(s) a node can be reached,

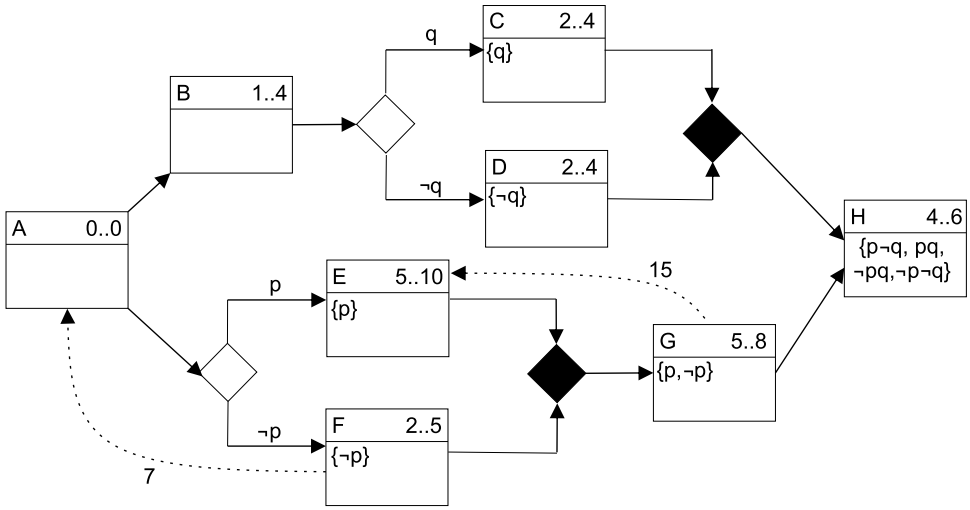

Fig. 1 Process graph with labels

i.e., which decisions were taken to reach a node. In contrast to [8], we define a label as a set of different possible decisions which can lead to the considered node. We formally define the basis for representing these paths and for comparing these paths below.

Definition 1 (*Propositional Terms*) Let L be a set of propositional letters. Let $p \in L$: then p and $\neg p$ are literals, $\mathcal{L}$ is the set of all literals over L. Let $l_1, \dots l_n$ be literals, then $l_1 \wedge \dots \wedge l_n$ are conjunctive terms.

We call t a minterm over a set of letters L, if t is a conjunctive term over the set of literals $\mathcal{L}$ and for each $p \in L$ either p or $\neg p$ appears in t. $\mathcal{M}(L)$ is the set of all minterms over $\mathcal{L}$.

Two terms t_1, t_2 are compatible ($t_1 \simeq t_2$), if there exist a minterm $t \in \mathcal{M}(\mathcal{L})$ such that $t \rightarrow t_1 \wedge t \rightarrow t_2$.

Two sets of conjunctive terms T_1 and T_2 are compatible, ($T_1 \simeq T_2$), iff $\forall t_1 \in T_1\ \exists t_2 \in T_2$ with ($t_1 \simeq t_2$), and $\forall t_2 \in T_2\ \exists t_1 \in T_1$ with ($t_1 \simeq t_2$).

Let $T = T_1, \dots T_n$ be sets of conjunctive terms over $\mathcal{L}$. We define the cross conjunction of T as $\bigotimes T = \{t_1 \wedge \dots \wedge t_n | t_i \in T_i, 1 \leq i \leq n\}$.

We assign a unique propositional letter p to each XOR-split and adorn one outgoing edge of the split node with p, the other with $\neg p$. The labels, we assign to the nodes, are sets of conjunctive terms over the propositional letters of predecessors of these nodes. Each term in the label of a node represents a different path to reach this node. In other terms it represents all possible combinations of decisions at XOR-split nodes leading to this node. In the following we define the representation of the model in form of a process graph formally.

Definition 2 (*Process Graph*) Let N be a set of nodes, $E \subseteq N \times N$ a set of edges, $B \subseteq N \times N \times \mathbb{R}$ a set of temporal constraints, and L a set of propositional letters. $P(N, E, B, L)$ is a *temporally constrained process graph*, iff (N, E) forms a connected directed acyclic graph.

A node n has the following properties:

Type: $n.type \in \{node, xs, xj\}$, where xs represents Xor-split and xj Xor-join.

Label: $n.L \in \mathcal{P}(\mathcal{L})$

Duration: $n.d_{min}$ (minimum duration) and $n.d_{max}$ (maximum duration).

An edge $e \in E$ has the property $e.l \in \{True\} \cup L \cup \{\neg l | l \in L\}$.

Constraints: B consist of upper-bound constraints, (s, d, δ) representing the constraint $d \leq s + \delta$, where $s \in N$ is called the source and $d \in N$ is called the destination node, and $\delta \geq 0$ is some real.

The function $x.l$ assigns a unique propositional letter $p \in L$ to each Xor-split node $x \in N$. For each Xor-split node x there are two outgoing edges e_1 and e_2 in E with $e_1.l = x.l$ and $e_2.l = \neg x.l$. All other edges have the label *True*.

A process graph has 1 *start* node, which has no predecessor and at least 1 *stop* node which has no successor. There is a path from the start node to every stop node. Every node is on a path from the start node to a stop node. A node of type xs has exactly 2 successor nodes, a node of type xj can have one or two predecessor nodes.

E^+ denotes the transitive closure of E. For a node n, $n.Pred$ denotes the set of all direct predecessors of n: $n.Pred := \{m | \exists (m, n) \in E\}$, and $n.Succ$ the set of all direct successors of n: $n.Succ := \{m | \exists (n, m) \in E\}$. $n.Pred^+$ denotes the set of all (direct or indirect) predecessors of n: $n.Pred^+ := \{m | \exists (m, n) \in E^+\}$, and $n.Succ^+$ is the set of all successor of n: $n.Succ^+ := \{m | \exists (n, m) \in E^+\}$.

The label of a node n is defined as (1) $n.L = \{True\}$, if n is a start node; (2) $n.L = \bigcup_{m \in n.Pred} m.L$, if $n.type = xj$; (3) $n.L = \bigotimes \{m.L | m \in n.Pred\}$ for any other node. □

It is easy to see that every well formed workflow net [34] can be represented as process graph according to the definition above. Figure 1 shows an example for a process graph with labels. The activities are represented as rectangles with their names, minimum and maximum duration, and labels. White diamonds represent XOR-split nodes and black diamonds XOR-join nodes. Upper-bound constraints $ubc(s, d, \delta)$ are shown as dotted arcs from destination d to source node s with the label δ.

To avoid questionable process definitions with impossible paths, ill defined joins and void constraints we define requirements for well-formed process graphs: for simple-merge XOR-joins it must not happen that both of its predecessor are executed in a single instance, i.e. the predecessors of an XOR-join node must have disjoint labels. If a node has several predecessors (implicit And-join) all these predecessors have equivalent labels. For all nodes except (except XOR-join nodes) we require that their predecessors are not mutually exclusive, i.e., False must not appear in labels of nodes. The stop nodes are mutually exclusive. And finally, for source and destination nodes of an upper bound constraint there has to be the possibility for an instance of the process in which both are activated - requiring that their labels are compatible.

Definition 3 (*Well-Formed Process Graph*) A process graph $P(N, E, B, L)$ is well-formed, iff

(1) $\forall n, n_1, n_2 \in N : n.type = xj, n_1 \neq n_2, (n_1, n), (n_2, n) \in E \Rightarrow n_1.L \wedge n_2.L \rightarrow False$, and
(2) $\forall n \in N \; \forall p \in n.L : p \neq False$, and
(3) $\forall s, s' \in N : \nexists (s, n) \in E : \forall t \in \mathcal{M} \Leftarrow \mathcal{L} \Rightarrow : t \rightarrow s.L \Rightarrow \neg(t \rightarrow s'.L)$, and
(4) $\forall (s, d, \delta) \in B : \exists t \in \mathcal{M}(L) : t \rightarrow s.L, t \rightarrow d.L$

In the following, we assume that all process graphs are well-formed. Below we formalize some properties of well formed process graphs which we will need later. The first observation is that the terms in the label of a node are pairwise disjoint.

Lemma 1 *For a well-formed process graph $P(N, E, B, L)$, $\forall n \in N, \forall p_1 \neq p_2 \in n.L : \neg(p_1 \wedge p_2)$.*

Proof Follows from Definitions 2 and 4. □

The next lemma states that for all nodes except XOR-join nodes, the label implies the label of the predecessor nodes.

Lemma 2 *For a well-formed process graph $P(N, E, B, L)$, $\forall n \in N, n.type \neq xj \; \forall p \in n.L \; \forall m \in n.Pred : p \rightarrow m.L$.*

Proof Follows from Definition 4(2): if the labels of the predecessor of a node are incompatible, then the cross product of their labels would contain *False*, which is a contradiction to P being well-formed. □

We define 2 nodes as parallel if they can be both in one run, but neither is successor of the other.

Definition 4 (*Parallel*) Two nodes $n, m \in N$ a well-formed process graph $P(N, E, B, L)$ are parallel ($n||m$), iff $n \notin m.Succ^+, m \notin n.Succ^+, \exists t \in \mathcal{M} \Leftarrow \mathcal{L} \Rightarrow : (t \rightarrow n.L) \wedge (t \rightarrow m.L)$.

Lemma 3 *For a well-formed process graph $P(N, E, B, L)$, $\forall n, m \in N$ with $n||m$ the following holds*:

(1) $\exists a \in N, a.type \neq xs, n, m \in a.Succ^+$,
(2) $\forall p \in n.L \exists q \in m.L : p \simeq q$,
(3) $\forall r \in a.L, \forall p \in n.Lp \rightarrow r, \forall q \in m.L : p \rightarrow r \wedge q \rightarrow r \Rightarrow p \simeq q$.

Proof

(1) If n and m are not successors in one way or the other, and they are not mutually exclusive, they they must have a common ancestor, which is not an XOR-split node, since all nodes are successors of the start node.
(2) Definition 4(3) requires that both have a common successor, which is not an XOR-join, and then, the lemma follows from Lemma 2.
(3) The immediate successors of a node which is not an XOR-split-node have the same labels. Parallel successors of a node can only have those propositional letters in their labels in common, which are already in the letter of their common predecessor (there cannot be the same XOR-split node between n resp. m and their closest common predecessor node. Hence, (3) follows from Lemmas 1 and 2. □

The lemma above states that if 2 nodes are not successors or mutually exclusive then they can both appear in a run independent of the path by which they are reached.

Schedules and Controllability

Now, we can define the semantics of the temporal constraints by defining which possible execution scenarios (traces with time stamps) are considered as correct. Then we define schedules as a definition of admissible intervals for the start and end events of the nodes in a process graph. Based on these definitions, we can define the properties of controllability and dynamic controllability as notions of the correctness of a process definition with temporal constraints.

Definition 5 (*Scenario*) A scenario $\bar{S}$ for a process $P(N, E, B, L)$ associates each $n \in N$: with 2 timestamps t_s and t_e, the time points of the start and end events of a process instance.

Definition 6 (*Valid Scenario*) A scenario $\bar{S}$ for a process $P(N, E, B, L)$ is valid, iff the following constraints hold: $\forall n, m \in N$

(1) $n.t_s + n.d_{min} \leq n.t_e \leq n.t_s + n.d_{max}$,
(2) $n \in m.Pred^+ \Rightarrow n.t_e \leq m.t_s$, and
(3) $(n, m, \delta) \in B \Rightarrow m.t_e \leq n.t_e + \delta$.

A schedule defines execution intervals for activities.

Definition 7 (*Schedule*) A schedule S for a process $P(N, E, B, L)$ associates each $n \in N$ with (F_s, T_s) and (F_e, T_e), execution intervals for start and end events. We write $(n, (F_s, T_s), (F_e, T_e)) \in S$ for a schedule entry.

In this definition F_s (*From start*) represents the earliest time point for starting node n, T_s (*To start*) the latest time point for starting n, and F_e (*From end*) the earliest time point for finishing n, and finally T_e (*To end*) the latest time point for finishing n.

The property of controllability of a process requires that there is a schedule for the process, such that all scenarios are valid, for which the time-stamps of the scenarios are taken from the respective intervals of this schedule. In the following, we will call a schedule with such a property *controllability schedule*.

Definition 8 (*Controllability*) A process $P(N, E, B, L)$ is controllable, iff it has a schedule S (controllability schedule), such that each scenario $\bar{S}$ for P is valid, if $\forall n\ n.F_s \leq n.t_s \leq n.T_s$ and $n.F_e \leq n.t_e \leq n.T_e$.

This definition of controllability leads to rather rigid schedules as controllability schedules, in which the start of each activity is fixed to a single point in time instead of a time interval. In practice, for some activities, an interval is expected. Nevertheless, we follow here the usual definitions, which are sufficient for theoretical considerations on controllability and conditional controllability. This more rigid definition is also easier and does not cause a loss of generality. However, we consider that the definition of controllability can be relaxed to allow more flexible schedules without sacrificing the notion and the check-ability of controllability, and our definition of schedule already cares for these less rigid definitions.

Lemma 4 *In each controllability schedule $n.F_s = n.T_s$ has to hold for all nodes n.*

Proof If n starts at T_s then it has to finish before $T_s + n.d_{max}$; if it starts at F_s, then it could finish at $F_s + d$ or later. Hence, $n.d \leq F_e - T_s \leq T_e - F_s \leq n.d_{max}$, or $n.T_s + n.d_{max} \leq n.T_e \leq n.F_s + n.d_{max}$ can only be satisfied if $n.T_s = n.F_s$ and $n.T_e = n.F_s + n.d_{max}$ has to hold, which is only possible, if $T_s = F_s$ □

Next, we define, when we regard a schedule as correct. As might be expected, we are able to proof, that correct schedules imply controllability.

Definition 9 (*Correct Schedule*) A schedule S of a process $P(N, E, B, L)$ is correct, iff $\forall n, m \in N$, $\forall (n, (F_s, T_s), (F_e, T_e)) \in S, \forall (m, (F'_s, T'_s), (F'_e, T'_e)) \in S$:

(1) $F_e = T_s + n.d_{min}$,
(2) $T_e = F_s + n.d_{max}$,
(3) $F_e + n.d_{max} - n.d_{min} \leq T_e$,
(4) $n \in m.Pred \Rightarrow T_e \leq F'_s$,
(5) $(n, m, \delta) \in B, n.L \simeq m.L \Rightarrow T'_e \leq F_e + \delta$.

In this definition, the last item requires that the end event of the destination node m is at most δ time units after the end event of the source node n, if there is an instance type which contains both n and m. This is the case, if the labels of n and m are not contradictory ($n.L \simeq m.L$). If no instance type contains both source and destination node (their labels are contradictory), then the upper-bound constraint is trivially satisfied.

Lemma 5 (Controllability Schedule) *A process is controllable, iff it has a correct schedule.*

Proof That a correct schedule fulfills all requirements of a controllability schedule follows immediately from the definitions. In the other direction: it is easy to see that each controllability schedule is correct. □

All time points and durations in the definition of a temporally constrained process are always relative to the process start. If a schedule exists for a process, we can shift the schedule as long as we maintain the relative position of each event, i.e. we can add an integer to all start- and end-times. Therefore, if a schedule exists, actually a manifold of schedules exist, in particular one, where the start time of the process is 0.

Lemma 6 (0-Schedule) *If there is a correct schedule S for $P(N, E, B, L)$, then for each $n \in N$ there is a correct conditional schedule S_n^0 with $n.F_s = n.T_s = 0$.*

Proof Let S be a correct schedule, let $n \in N$, let $(n, (\tau_s, \tau_s), (\tau', \tau'')) \in S$. Then, S_n^0 is derived from S by adding $-\tau_s$ to all From and To values in S. S_n^0 is a correct schedule, because adding a scalar τ is an equivalence transformation for the inequalities of Definition 9 and $n.Fs = n.Ts = 0$. □

With this lemma, we know that it is sufficient to check whether a schedule exists in which the start node of the process starts at time point 0.

Computation of Schedules

In this section we combine the checking, whether a process is controllable, with the actual computation of a correct schedule for a process. For the rest of this section we assume that a process has a deadline (Ω), i.e., an upper-bound constraint from the start node to all end nodes of the process. In [14] we presented an algorithm without this assumption, which splits up a process graph into cliques (strong connected components), computes a schedule for each clique and then stitches together a complete schedule from these component schedules. In this paper, we essentially assume the whole process graph is one clique by assuming a deadline. However, with reference to [14], we claim that this is without loss of generality.

For computing a schedule for a temporally constrained process we use an intermediary structure called schedule frame (formerly also known as (unfolded) timed graph [14, 18, 19]). The schedule frame of a process P has the same structure as P, but has additional attributes for each node.

Figure 2 shows the schedule frame for the process of Fig. 1. Activities are represented as rectangles with their activity name (e.g., G, their minimum and maximum duration (e.g. 5..8) and their labels (e.g., $\{p, \neg p\}$. XOR-split nodes are represented as diamonds, XOR-join nodes as shaded diamonds. Upper-bound constraints are represented as dashed arrows from destination to source. Each node shows the name of the activity, and the minimum and maximum duration in the top line. Below the values for $E_b, L_b, E_w, and L_w$ are given.

In a schedule frame we consider intervals for the end event of each node x for each term in $x.L$. We will show later that all entries of a 0-schedule have to be within these intervals. The intervals are defined in terms of E- and L-values. A node n of the schedule frame cannot finish before $n.E_b$ (Earliest time point best case), if all contingent activities up to n finish at their minimum duration. It cannot finish before $n.E_w$ (Earliest time point worst case), if they take their maximum duration. The L-values represent time points when a node has to finish at the latest, to satisfy all temporal constraints: if n finishes before $n.L_b$ all constraints are satisfied, if all succeeding contingent activities only take their minimum duration; if n finishes before $n.L_w$ all constraints are satisfied, even if all succeeding activities use their maximum duration.

Definition 10 (*Schedule Frame*) $U(N, E, L, B)$ is a schedule frame for a process $P(N, E, L, B)$ with each $n \in N$ associated with intervals for end events of activities: E_b, E_w, L_w, L_b. A schedule frame is correct , iff $\forall n \in N, \forall m \in n.Pred$:

(1) $n.E_w \leq n.L_w$,
(2) $m.E_w + n.d_{max} \leq n.E_w$,
(3) $m.E_b + n.d_{min} \leq n.E_b$,
(4) $m.L_w + n.d_{max} \leq n.L_w$,
(5) $m.L_b + n.d_{min} \leq n.L_b$,
(6) $\forall (s, d, \delta) \in B, s.E_b + \delta \leq d.E_w$,
(7) $\forall (s, d, \delta) \in B, s.L_w + d \leq d.L_b$

We now analyze the relationship between a schedule frame and a schedule. It is easy to see that a schedule frame is more general than a schedule. However, as we

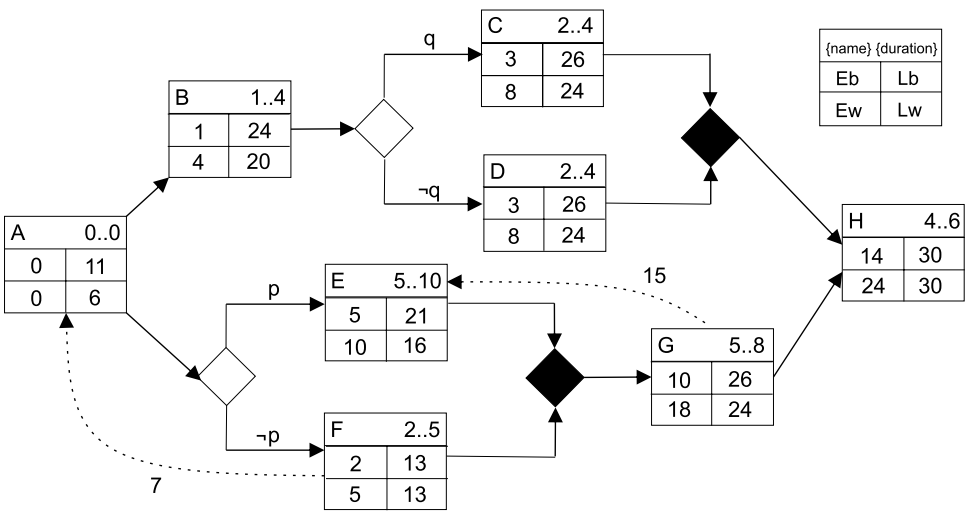

Fig. 2 Correct schedule frame

describe precisely in the next Lemma, processes either have both or none.

Lemma 7 *A temporally constrained process graph has a correct schedule, iff it has correct schedule frame.*

Proof Algorithm 1 (CompS) computes a schedule from a correct schedule frame. We can show that this schedule frame is correct by checking all conditions of Definition 9 assuming the conditions of Definition 10 to hold. On the other hand, the values of a correct schedule can be immediately used for a correct schedule frame by setting $n.E_b = n.E_w = n.F_e$ and $n.L_b = n.L_w = n.F_s$ for each n. □

Algorithm 1 CompS(U) Compute schedule from a correct schedule frame

1: {Input: $U(N, E, L, B)$ correct schedule frame}
2: {Output: correct schedule S}
3: s :=start node in U
4: {Schedule entries for start nodes}
5: $s.F_s := s.E_b - s.d_{min}; s.T_s := s.F_s$
6: $s.F_e := s.E_b; s.T_e := s.F_e + s.d_{max} - s.d_{min}$
7: **for all** $n \in N - \{s\}$ in a topological order **do**
8: $n.F_s := max(\{n.E_b - n.d_{min}\} \cup \{m.T_e | m \in n.Pred\})$
9: $n.T_s := n.F_s$
10: $n.F_e := n.F_s + n.d_{min}$
11: $n.T_e := n.T_s + n.d_{max}$
12: **end for**
13: **return**

The procedure of computing a correct schedule frame (Algorithm 2) consists of the following steps. First, we initialize the temporal values of the schedule frame: the E values of all nodes are set to 0, and the L values of all nodes are set to the deadline Ω (for $ubc(start, end, \Omega)$). Then we compute the schedule frame according to the structural constraints defined by the topology of the graph by forward calculation of E values and backward calculation of L values. Then we check all upper-bound constraints. If an upper-bound constraint is violated, we incorporate it by increasing the E values of the source node and/or the L values of the destination node. This procedure is repeated, until all constraints are satisfied, or there is a node n, for which the invariant $n.E < n.L$ is violated and hence no solution exists. This algorithm is an adaption of the algorithms presented in [19] to also deal with contingent activities.

Algorithm 2 CompTG(U, Ω) returns: Boolean – Compute correct schedule frame U with deadline Ω

1: {Input: $U(N, E, L, B)$ is a schedule frame}
2: {Output: return True and correct schedule frame or return False}
3: ok := False
4: **for all** n inN **do**
5: $n.E_b, E_w := 0; L_b, L_w := \Omega$
6: **end for**
7: **while** not ok **do**
8: ok:= true
9: **for all** $n \in N$ in a topological order **do**
10: {forward calculation}
11: $n.E_b := max(\{n.E_b\} \cup \{p.E_b + n.d_{min} | p \in n.Pred\})$
12: $n.E_w := max(\{n.E_w, n.E_b + n.d_{max} - n.d_{min}\} \cup \{p.E_w + n.d_{max} | p \in n.Pred\})$
13: **if** $n.L_w < n.E_w$ **then**
14: **return** False
15: **end if**
16: **end for**
17: **for all** $n \in N$ in a reverse topological order **do**
18: {backward calculation}
19: $n.L_w := min(\{n.L_w\} \cup \{s.L_w - s.d_{max} | s \in n.Succ\})$
20: $n.L_b := min(\{n.L_b\} \cup \{s.L_b - s.d_{min} | s \in n.Succ\})$
21: **if** $n.L_w < n.E_w$ **then**
22: **return** False
23: **end if**
24: **end for**
25: **for all** $(s, d, \delta) \in B$ **do**
26: {incorporation of upper-bound constraints}
27: **if** $\delta < (d.E_w - s.E_b)$ **then**
28: ok:= false
29: $s.E_b := max(s.E_b, d.E_w - \delta)$
30: $s.E_w := max(s.E_w, s.E_b)$
31: **end if**
32: **if** $\delta < (d.L_b - s.L_w)$ **then**
33: ok:= false
34: $d.L_b := min(d.L_b, s.L_w + \delta)$
35: $d.L_w := min(d.L_w, d.L_b)$
36: **end if**
37: **end for**
38: **end while**
39: **return** True

Figure 2 shows the schedule frame for the our example in Fig. 1 with the E and L values of each node after CompTG finished successfully.

The next theorem states that the CompTG procedure is sound and complete, which means that if there is a correct schedule for a temporally constrained process then the procedure will compute a correct schedule and return TRUE, and if there is no correct schedule for the process it will return FALSE.

Theorem 1 *CompTG* (*Algorithm* 2) *is sound and complete in the sense that* (*a*) *if it returns TRUE then the computed schedule frame is correct, and* (*b*) *if it returns FALSE then there is no correct schedule frame.*

Proof Soundness can be easily shown as in CompTG all conditions for a correct schedule frame are either checked explicitly in the algorithm or are ensured by the assignments in the forward and backward calculations. □

For showing completeness we refer to Lemma 6 and the overall deadline Ω such that no correct schedule entry can have a value outside the interval $[0, \Omega]$ can be valid. Hence the schedule frame is correctly initialized. Furthermore, we observe that for every node every execution interval of a correct conditional 0-schedule has to be within the interval of E and L values with which the schedule frame is initialized. In addition, it is easy to show that the requirements of correctness for a conditional schedule are more strict than for a correct schedule frame. The forward resp. backward calculations set the E values resp. L values to the lowest resp. greatest values that satisfy the conditions (1) to (5) of Definition 10. The algorithm iteratively checks all temporal constraints. If an inequality in the definition of a constraint is violated the algorithm sets the E values to the smallest and L values to the greatest values such that this inequality is satisfied and thus computes the largest interval which avoids this violation. So if the algorithm cannot compute a correct schedule frame, there is no correct schedule frame (and hence also no correct conditional schedule). □

With this theorem, we can conclude that applying the CompTG procedure to a temporally constrained process with a deadline is a sound and complete algorithm for checking its controllability.

Conditional Schedules and Conditional Controllability

As discussed above, a process is controllable if it admits a schedule. This notion is known to be too restrictive [17, 27] and therefore, the notion of a history-dependent or conditional schedule was introduced [10, 14, 17], which offers the possibility to define the execution intervals for a node depending on past decisions or observations. In the definition of conditional controllability, we use here, the execution interval of a node might depend on the observed outcomes of XOR-splits preceding this node. There are other more general definitions of dynamic controllability [3, 8, 27], which allow all information (in particular start and end time of contingent activities), which temporally precedes the node, can be used by a node to define its allowed time interval.

Now, we can define the semantics of the temporal constraints by defining, which possible execution scenarios (traces with time stamps) are considered as correct. A conditional scenario takes into account that not every node of the process appears in every process instance due to XOR-splits. The possible combinations of nodes are determined by any possible combination of decisions which are in turn represented by all minterms over the propositional letters of the process definition. In such a conditional scenario a node n can have different time points for different minterms. A conditional scenario is valid, if each of its scenario projections (defined by a particular minterm) is a valid scenario.

Definition 11 (*Conditional Scenario*) A conditional scenario $\bar{S}$ for $P(N, E, L, B)$ associates for each $t \in \mathcal{M}(L)$ each $n \in N$ with $t \to n.L$ with 2 timestamps t_s and t_e, the time points of the start and end events of a process instance. We call $(t, n, t_s, t_e) \in \bar{S}$ a scenario entry.

$\bar{S}$ is valid, iff $\forall t \in \mathcal{M} \Leftarrow \mathcal{L} \Rightarrow \forall (t, n, nt_s, nt_e), (t, m, mt_s, mt_e) \in \bar{S}$

(1) $nt_s + n.d_{min} \leq nt_e \leq nt_s + n.d_{max}$,
(2) $n \in m.Pred^+ \Rightarrow nt_e \leq mt_s$, and
(3) $\forall (n, m, \delta) \in B : mt_e \leq nt_e + \delta$.

In a *conditional schedule*, the execution interval for a node n in the process graph may depend only on the decisions taken before the execution of n. The decisions are observed by monitoring which outgoing edge of XOR split nodes preceding n were taken, resp. which successors of an XOR-split were enabled. Therefore, different paths to a node n might lead to different execution intervals for n. The label of a node n in a process graph P exactly contains the different possibilities for reaching this node. So in a conditional schedule we assign (possibly different) execution intervals to each node for each of the terms in its label. As it is possible to assign the same execution interval for different paths leading to a node, we use a more compact representation and allow the disjunction of label elements in the definition of schedule entries.

Definition 12 (*Conditional Schedule*) A conditional schedule S for a process graph $P(N, E, L, B)$ is a set of schedule entries $\{(n, q, (F_s, T_s), (F_e, T_e))\}$ which associate each $n \in N$ and a propositional term q with intervals for the start and end events of n, such that for each $n \in N$ and for each $p \in n.L \exists (n, q, (F_s, T_s), (F_e, T_e)) \in S$ with $p \to q$ and for each $(n, q, (F_s, T_s), (F_e, T_e)) \in S$ there is a set $\{p_1, \ldots, p_i\} \subseteq n.L$ with $q \equiv p_1 \vee \cdots \vee p_i$.

F_s (say *From start*) represents the earliest time point for starting node n, T_s (*To start*) the latest time point for starting n, and F_e (*From end*) the earliest time point for finishing n, and finally T_e (*To end*) the latest time point for finishing n. The definition of conditional controllability requires that a conditional schedule exists such that each scenario is valid, if its time points are within the limits of the execution intervals defined in the conditional schedule.

Definition 13 (*Conditional Controllability*) A process graph $P(N, E, L, B)$ is conditionally controllable, iff it has a conditional schedule S such that each conditional scenario $\bar{S}$ is valid, if for each minterm t for each scenario entry $(t, n, t_s, t_e) \in \bar{S}$ for $(n, p, (F_s, T_s), (F_e, T_e)) \in S$ with $t \to p$: $F_s \leq t_s \leq T_s$ and $F_e \leq t_e \leq T_e$.

This definition is well formed since each minterm t implies at most one term in the label of a node, as all terms in a label are pairwise disjoint (Lemma 1).

Definition 14 (*Correct Conditional Schedule*) A conditional schedule S of a process $P(N, E, L, B)$ is correct, iff $\forall n, m \in N$, $\forall (n, p, (F_s, T_s), (F_e, T_e)) \in S$, $\forall (m, q, (F'_s, T'_s), (F'_e, T'_e)) \in S$ with $p \simeq q$

(1) $F_e = T_s + n.d_{min}$,
(2) $T_e = F_s + n.d_{max}$,
(3) $F_e + n.d_{max} - n.d_{min} \leq T_e$,
(4) $n \in m.Pred^+ \Rightarrow T_e \leq F'_s$,
(5) $(n, m, \delta) \in B, p \simeq q \Rightarrow T'_e \leq F_e + \delta$.

Lemma 8 *A process is conditionally controllable, iff it has a correct conditional schedule.*

Proof From Definitions 12, 13, and 14 we can easily see, that a correct conditional schedule fulfills all requirements of a controllability conditional schedule. In the other direction: each controllability conditional schedule is correct as it satisfies all conditions of Definition 14. □

If a process admits a correct conditional schedule, then for all nodes n it also admits a correct schedule where the node n starts at time point 0.

Lemma 9 (Conditional 0-Schedule) *If there is a correct conditional schedule S for $P(N, E, L, B)$, then for each $n \in N$ there is a correct conditional schedule S^0_n with $n.F_s = n.T_s = 0$.*

Proof Let S be a correct conditional schedule for a process $P(N, E, L, B)$, let $n \in N$, let $(n, p, (\tau_s, \tau_s), (\tau', \tau'')) \in S$. Then ,$S^0_n$ is derived from S by adding $-\tau_s$ to all From and To values in S. S^0_n is a correct conditional schedule since addition of scalars is an equivalence transformation for the inequalities of Definition 14 and $n.Fs = n.Ts = 0$. □

The major difference between a schedule and a conditional schedule is that a conditional schedule can assign different start times to a node in the process graph, depending on the different paths by which this node can be reached.

We now define an unfolded process graph (shown in Fig. 3) which separates all these paths by duplicating a node, if it can be reached by several paths. The unfolded process graph is equivalent to the original process graph in that it admits exactly the same set of traces as the original graph but separates the possible traces.

In an unfolded process each node of the source process has a copy for each term in its label and the edges are wired accordingly, if the predecessor has a compatible label.

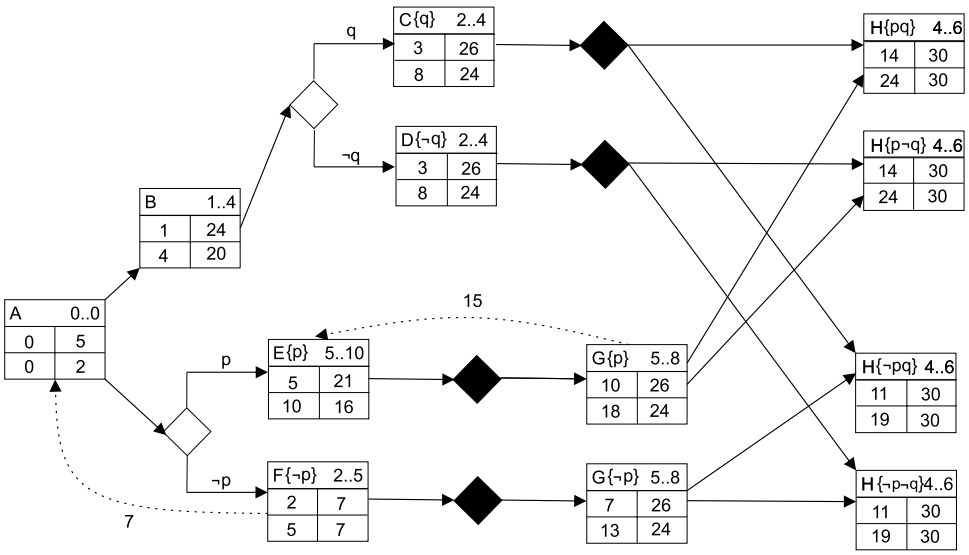

Fig. 3 Unfolded process graph with correct schedule

In an unfolded graph each XOR-join node has exactly 1 predecessor.

Definition 15 (*Unfolded Graph*) $U(N, E, L, B)$ is an unfolded processes graph for a process $P(N', E', L, B')$, iff $N = \{(n, l) | n \in N', l \in n.L\}$, $E = \{((n, l), (m, k)) | (n, m) \in E', l \simeq k\}, B = \{((n, l), (m, k), \delta) | (n, m, \delta) \in B', l \simeq k\}$.

Essentially, an unfolded process graph can be constructed by duplicating all XOR-join nodes, which have more than one predecessor, and all their successors.

Lemma 1 *If $P(N', E', L, B')$ is a well-formed process, then its unfolded process $U(N, E, L, B)$ is a well formed process.*

Proof Let a process graph $P(N', E', L, B')$ be well formed. Its unfolded process graph is well formed (Definition 4) as

(1) in U each xj-node has only 1 predecessor
(2) $\forall (n, p) \in N, (n, p).L = \{p\} and p \neq False$, since P is well formed.
(3) $\forall ((s, p), (d, q), \delta) \in B : \exists t \in \mathcal{M}(L) : t \rightarrow s.L, t \rightarrow d.L$ holds since $p \simeq q$ according to Definition 15. □

Now we can show the relationship between unfolding, schedules and conditional schedules:

Theorem 2 *A process graph is conditionally controllable, iff its unfolded graph is controllable.*

Proof Let $P(N', E', L, B')$ be process,and $U(N, E, L, B)$ its unfolded process. If P is conditionally controllable, it has a conditional schedule S'. It is easy to see that $S = \{((n, p), (F_s, T_s), (F_e, T_e)) | (n, p, (F_s, T_s), (F_e, T_e)) \in S\}$ is a correct schedule for U, hence U is controllable. The other direction follows in analogy. □

Figure 3 shows the schedule frame for the unfolded process graph of Fig. 1. There are three different occurrences of node H, as there are 4 different ways how node H can be reached. Each occurrence is described by a different term in the label of node H. The E and L values of the different occurrences of node H are different which leads to a conditional schedule, where the start of an activity depends on the path by which this node is reached. However, not all occurrences of H have different values are different, a property, we will exploit below.

Problem Complexity

In this section, we study the complexity of the problem of computing conditional schedules by analyzing the possible size of a conditional schedule for a given process model. We formalize and show below that in the worst case the size of a schedule is exponential in the number of XOR-splits.

Theorem 3 (Problem Complexity) *Let $P(N, E, L, B)$ and let x be the number of XOR-split nodes in N ($x = |\{n \in N | n.type = xs\}$). The minimum size of a conditional schedule for P is $|N|$ and the maximum necessary size is $|N| * 2^x$.*

Proof For the minimum size: Any conditional schedule contains at least one schedule entry for each activity. Actually, if a process is controllable, then one schedule entry for each node is also sufficient, making the number of nodes the minimum size of a conditional schedule. □

For the maximum necessary size: The number of different possible propositional terms in the labels of the nodes is 2^x for $x = |\{n \in N | n.type = xs\}$, a conditional schedule cannot be larger than $|N|^x$. We show that the size can be exponential in the number of XOR-splits by constructing a process (*proof-process*) with x XOR-splits with a set of temporal constraints and show that in its only schedule there is an activity which has 2^x different disjoint scheduling intervals.

Figure 4 shows the pattern for this process. It consist of an activity A followed by n XOR-blocks numbered from n to 1 followed by activity D. Each XOR-block $X_i, 1 \leq i \leq n$ consists of two alternate activities B_i and C_i. The min- and max-durations of the nodes are as follows: $B_i.d = B_i.x = 1$ and $C_i.d = C_i.x = 2^{i-1} + 1$ for all i from 1 to $n, D.d = D.x = 1$. All other nodes have duration 0. We define the following upper-bound constraints: $\{(B_{i+1}, B_i, C_i.d), (C_{i+1}, B_i, 1), (Bi + 1, Ci, 1), (C_{i+1}, C_i, C_{i+1}.d) | 1 \leq n - 1\}$, $(A, B_n, 1), (A, C_n, C_n.d_{min}), (B_1, D, 1), (C_1, D, 1)$. After some calculations it is easy to see that there is only one possible conditional schedule and in this conditional schedule there are 2^n non-overlapping intervals for activity D: $[n + 1, n + 1] \ldots [2^n + n]$. Figure 5 shows the schedule for $n = 3$. □

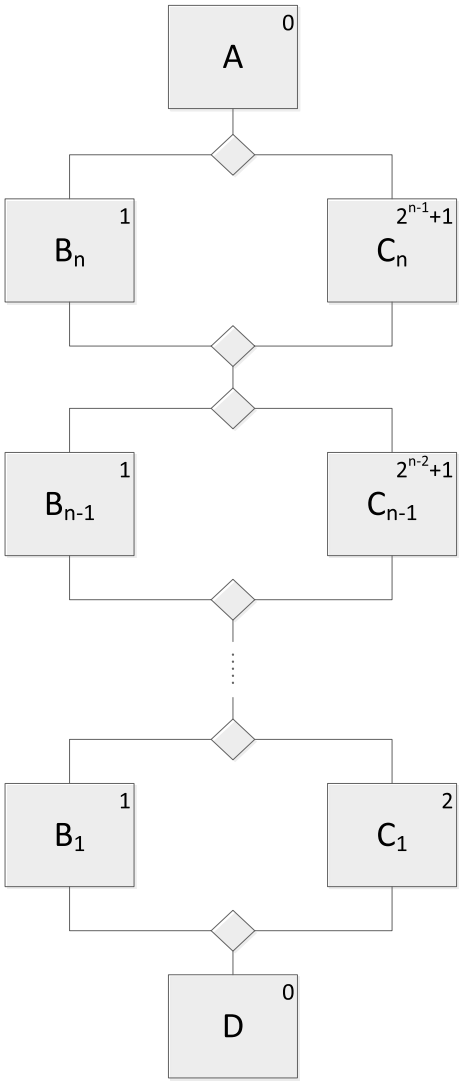

Fig. 4 Pattern for the proof-process

Efficient Computation of Conditional Schedules

In a straightforward way we can exploit Theorem 2 to realize a procedure to check the conditional controllability of a process and to compute a conditional schedule: unfold the process graph and then compute a schedule for the unfolded process with the presented algorithm. It is easy to see that this is a sound, complete, and effective procedure. The disadvantage of this procedure is, that the complexity of this algorithm is always exponential. In the last section we showed that indeed such as a procedure has to be exponential in the number of XOR-split nodes in the worst case. However, there are more efficient procedures for average cases. In the best case, i.e., if the process is even strictly controllable, the size of a conditional schedule is linear in the number of nodes. Therefore, we aim at more efficient ways for computing a conditional schedule than by computing a schedule for the original rather than the unfolded process.

Essentially, our strategy to improve the algorithm aims at reducing the unfolding of the process graph, thus reducing the size of the schedule frame for the computation using partially unfolded graphs. We apply the following strategies:

The size of the schedule frame can be reduced, if not all nodes are unfolded, but only those relevant for incorporation of upper-bound constraints.

The goal of unfolding is to separate those paths of a process graph which are affected by the incorporation of an upper-bound constraint from those which are not. When we incorporate an upper-bound constraint, we change E- and L-values of nodes, which affect the values of other nodes. We unfold to separate those nodes that have to be affected by the change of values from those which need not be changed.

In a partially unfolded process graph a node combines potentially several nodes of the unfolded processes graph. So there is not a copy of a node for each term in the label of the node, but the labels are partitioned into disjoint subsets of terms and there is a copy for each partition.

Definition 16 (*Partially Unfolded Graph*) $U(N, E, L, B) = \Psi(P, N)$ is called a partially unfolded process graph for a process $P(N', E', B', L)$, iff

- $N = \{(n,p) | n \in N', p \subseteq n.L\}$,
  $\forall (n,p),(n,q) \in N : p = q \vee \neg(p \simeq q)$
- $E = \{((n,p),(m,q)) | (n,m) \in E', p \simeq q\}$,
- $B = \{((n,p),(m,q),\delta) | (n,m,\delta) \in B', p \simeq q\}$.

For defining a partially unfolded process graph, it is sufficient to define a partition of the label of each node. The edges and the constraints are then determined accordingly.

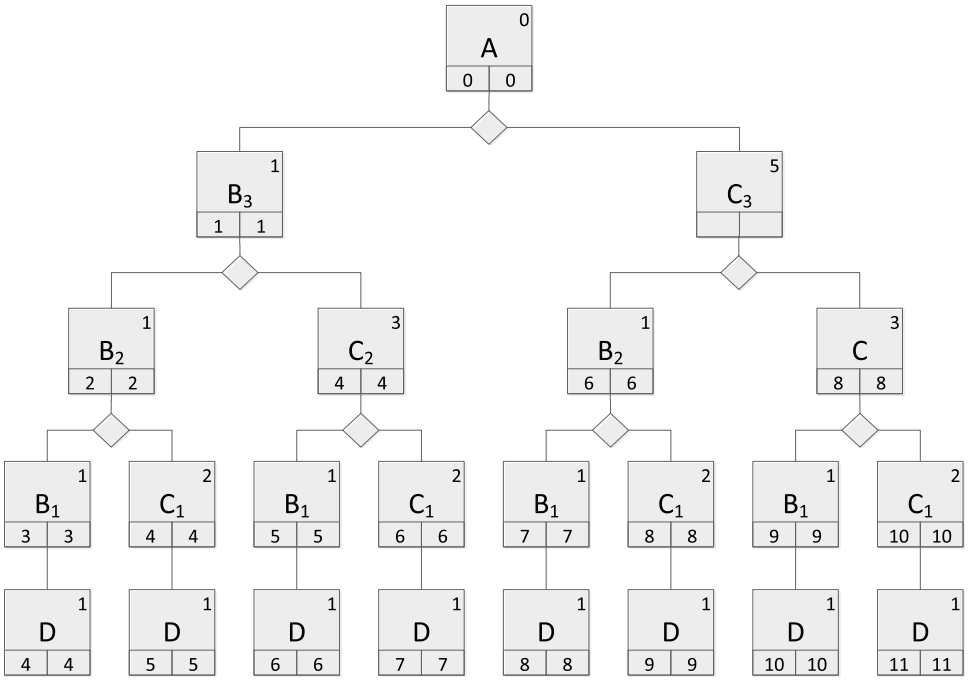

Fig. 5 Computed conditional schedule for the proof-process with the problem size $n = 3$

Algorithm 3 P-UNFOLD(U) Partial Unfolding of a schedule frame U

```
1: {U(N, E,L, B) is a partially unfolded schedule frame}
2: for all x ∈ {(n ∈ N|n.type = xj, ∃ubc(s, d, δ) ∈ B : n ∈ s.Succ+, ∃m ∈
   n.Pred, m ∉ s.Succ+} in a reverse topological order do
3:    {x is an XOR-join node with 2 predecessors x.Pred1 (s or successor of
      s) and x.Pred2 not successor of s}
4:    for all n ∈ N do
5:       copy[n] := n {initialize copy table}
6:    end for
7:    M := {x} ∪ x.Succ+
8:    for all m ∈ M do
9:       {duplicate all nodes after x including x}
10:      m' := CloneNode(m)
11:      copy[m] := m'
12:   end for
13:   {create edges to/from duplicated nodes}
14:   E := E ∪ {(copy[m], copy[n])|(m, n) ∈ E, (m ∈ M ∨ n ∈ M)}
15:   {The copies are not successors of s}
16:   E := E − {(x.Pred2, x), (x.Pred1, copy[x])}
17:   {migrate upper bound constraints}
18:   for all (s, d, δ) ∈ B|s ∈ M ∨ d ∈ M do
19:      if d ∈ s.Succ+ then
20:         if s ∈ M then
21:            B := B ∪ {(copy[s], copy[d], δ)}
22:         end if
23:      else
24:         a := ccp(s, d)
25:         {a is the closest common predecessor of s and d}
26:         if a ∈ M then
27:            B := B ∪ {(copy[s], d, δ), (s, copy[d], δ), (copy[s], copy[d], δ)}
28:         else
29:            B := B ∪ {(copy[s], copy[d], δ)}
30:         end if
31:      end if
32:   end for
33: end for
34: return
```

Which nodes have to be duplicated for a given upper-bound constraint $ubc(s, d, \delta)$? If we incorporate this upper-bound constraint, we change the E values of s and the L values of d. This in turn may cause changes in the predecessors of s and the successors of d. We now have to check which of these possible affected nodes in a partially unfolded graph could be duplicated, such that one copy is affected and the other not.

Let us first check the successors of s. Some of these nodes might be also reachable by paths which do not include s. And exactly these nodes form the unfold set of nodes, as they could be unfolded, such that their twin is not affected. Therefore, these nodes need to be duplicated and the partially unfolded graph has to contain 2 copies of these nodes: one which is a successor of s and one which does not have s.

If d is a successor of s also d might be included in these nodes and might be duplicated. If this is the case, only one copy of d is the destination node of the upper-bound constraint, the other not.

In the other direction: all predecessors of d are affected by a change of d's L values. It is easy to see, that these nodes can only be reached through nodes, which are also predecessors of d in an unfolded graph. Hence, the predecessors of d do not need to be considered for unfolding.

We formalize these considerations in the following theorem, which states that a process graph is sufficiently unfolded, if no successor of a source node s of an upper-bound constraint $ubc(s, d, \delta)$ is reachable through a path which does not include this source node s. Formally, this requires that each successor node of such s have only terms in their label which imply a term in the label of s.

Definition 17 (*Sufficient Unfold*) Let $P(N', E', L, B')$ be a process graph; $U(N, E, L, B) = \Psi(P, N)$ is a partially unfolded process graph. U is sufficiently unfolded, iff

(1) U is well formed.
(2) $\forall ubc(s, d, \delta) \in B\colon\ \forall p \in s.L\ \ \forall q \in d.L\colon\ (s, \{p\}) \in N$, $(d, \{q\}) \in N$.
(3) $\forall ubc(s, d, \delta) \in B$, $\forall n \in s.Succ^+\ \forall p \in n.L\ \exists q \in s.L\ :\ p \rightarrow q$.

The definition above requires (1) U to be well formed. This means in particular, that if a node is duplicated, also its successors are duplicated, as otherwise U would not be well formed (see Definition 4). It also requires a proper mapping of upper-bound constraints, i.e., an upper-bound constraint $ubc(s, d, \delta)$ is included, if all upper-bound constraints are between nodes which can actually appear in the same run, technically, if they are both induced by the same minterm over $\mathcal{M}\Leftarrow\mathcal{L}\Rightarrow$ (see Definition 4). Then the definition requires, that the source nodes of all upper-bound constraints are fully unfolded, i.e., for each term in their labels, there is a different node in the graph. The definition also requires (3) that each successor n of a source node s can only be part of a scenario, if the source node is also part of it. It also means that each minterm over L which implies $n.L$ also implies $s.L$. Therefore, this condition requires essentially that if the source node of an upper-bound constraint is unfolded, so are all its successors.

Theorem 4 *Let $P(N', E', L, B')$ be a process graph and $U(N, E, L, B) = \Psi(P, N)$ a sufficiently unfolded process graph.*

P is conditionally controllable, iff U is controllable.

Proof Let P(N",E",B",L) be a process graph, U'(N',E',B',L) a sufficiently unfolded graph, and let U(N,E,B,L) be a fully unfolded graph. □

With Theorem 2, we have to show that all copies of node of a sufficiently unfolded graph in an unfolded graph have the same values. I.e. $\forall n \in N'', \forall (n, Q) \in N' \forall q \in Q, \forall (n, q) \in N\ :\ (n, Q).E_b = (n, q).E_b$, etc.

The proof follows the argumentation above: the only condition for values of a correct schedule frame, in which values of a node do not depend on all of their immediate predecessors, their immediate successor, or their own duration are the conditions (6) and (7) of Definition 10, i.e. when the values are determined by an upper-bound constraint. Differences in the values of copies (n, p) and (n, q) of the same node $n \in N$ in the schedule frame are only possible, if n is source or destination of an upper-bound constraint. As the algorithm only changes the L values of a destination node, and these changes are propagated to the predecessors of a node anyhow, their unfolding would not change the values of these predecessors. Changes to the E values of a source node s of an upper-bound constraint, however propagates to all successors, even if they are reachable through different paths and therefore for runs which do not include S. Condition (3) of the definition requires, that all successors of a source node can only be reached through this source node, which eliminates the possibility above. □

With the theorem above, we now develop an algorithm for computing a sufficiently unfolded graph for an upper-bound constraint $ubc(s, d, \delta)$.

We observe that the label of a node is different from the label of its predecessor node only if it is (i) an XOR-join node (then the labels of its predecessors are disjoint), or (ii) the successor of a XOR-split node, where the label of the successor implies the label of the predecessor.

Each node, which has to be duplicated, has a predecessor XOR-join node, which also has to be duplicated. A node will be duplicated, if this preceding XOR-join node is duplicated. It is therefore sufficient to check for XOR-join nodes, which are successor of s and which have a direct predecessor, which is not a successor of s. We therefore define the unfold set for $s$as$\{(n, p) \in N | n.type = xj, (n, p) \in s.Succ^+, \exists (m, q) \in (n, p).Pred, (m, q) \notin s.Succ^+\}$ and unfold the graph at these nodes.

Algorithm 7 shows the procedure for partially unfolding a process graph. It unfolds the graph for all XOR-join nodes in the unfold set of all upper-bound constraints. Unfolding at a particular XOR-join node x duplicates all successors of x and adjusts the edges accordingly.

Now, we study the mapping of an upper-bound constraint $ubc(s, d, \delta)$ for the unfolding of one XOR-join node x, which duplicates some of the nodes. The nodes s, d, and their closest common predecessor a might each be duplicated, with s', d', a', respectively, as their possible twins with $s' \neq s$, $d' \neq d$, $a' \neq a$, or they are not duplicated which means that $(s' = s,\ d' = d,\ a' = a)$. The following cases have to be distinguished for the inclusion of additional upper-bound constraints:

(1) $s \neq s'$ and $d \neq d'$ and $d \in s.Succ^+$: $ubc(s', d', \delta)$
(2) $s = s'$ and $d \neq d'$, $d \in s.Succ^+$, and $d' \in s.Succ^+$: $ubc(s, d', \delta)$
(3) $d \notin s.Succ^+, a = a'$: $ubc(s, d', \delta), ubc(s', d, \delta), ubc(s', d', \delta)$
(4) $d \notin s.Succ^+, a \neq a'$: $ubc(s', d', \delta)$

Figure 6 shows the schedule frame for the partially unfolded process of Fig. 1 (for explanations see Fig. 2). The partial unfolding of a schedule frame leads to smaller graphs and hence to a faster computation of a correct schedule frame in comparison to a fully unfolded graph. Although

in this example, the difference to the fully unfolded schedule frame in Fig. 3 is not breathtaking, and of course, the reduction in size depends on the number and position of upper-bound constraints and XOR-split and XOR-join nodes. In the worst case, the partially unfolded graph is identical to the fully unfolded graph. In the next section, we empirically explore the possible gains.

Implementation and Evaluation

We implemented all the presented algorithms for full unfold, partial unfold (Algorithm 7), checking (conditional) controllability by computation of correct schedule frames (Algorithm 2) and computation of correct schedules (Algorithm 1) in Java. With these implementations, we performed experiments to evaluate the feasibility of the approach and, in particular, to explore potential improvements achieved by partial unfold versus full unfold.

We ran our experiments on a Windows 10 server with an Intel Xeon CPU with 16 cores and 2.2 GHz computation power and 132 GB of RAM. For the experiments we randomly generated 1600 processes with around 200 activities each and varying numbers of XOR constructs and upper-bound constraints. Also the placement of the defined number of XOR-splits and XOR-joins was generated randomly, as well as the defined number of upper-bound constraints.

We structured the test set into subsets according to their number of XOR-splits and UBCs. We specified a time-out after 30 min of computation, considering exceeding instances as not feasible. The following graphs summarize the figures collected by the experiments.

Figure 7 shows the size of the graphs in number of nodes for the full unfolded graph and the partial unfolded graph in relation to the number of XOR-splits. The experiments showed convincingly that the number of nodes of the partially unfolded graphs is significantly smaller.

Figure 8 shows the size of the graphs for different amounts of upper-bound constraints. The size of the fully unfolded graph only depends on the number and placement of XOR-splits and -joins. The slight variations are due to the randomly selected placement of these nodes. Apart from this, the size of the graph does not depend on the number of upper-bound constraints. In contrast, the number of nodes of partially unfolded graphs increases with the number of upper-bound constraints. However, even for very large numbers of upper-bound constraints (50 in our experiments), the size of the partially unfolded graphs is still 40% smaller than the size of the fully unfolded graphs.

Figure 9 documents the high performance gains achieved by applying the partial unfold algorithm in comparison to the full unfold. The performance figures are given for the different amounts of XOR constructs. This figure also shows that the approach is feasible for realistic sizes of process graphs. For 200 activities with 17 XOR constructs, the average time for computing a correct schedule frame did not exceed 40 s.

We can summarize the results of the experiments as follows: the presented approach for checking the conditional controllability of temporally constrained processes proved to be feasible for realistic sizes of processes and temporal constraints, in particular taking into account, that these are design time algorithms. In addition, the experiments show that applying partial rather than full unfold greatly reduces the size of the graphs and significantly improves the performance of the computation of conditional schedules.

Related Work

Our approach tries to contribute to the integration of time management in workflows, temporal consistency in service composition [24] and temporal constraint networks. There is already quite some body of research results in the area of temporal aspects of workflows and business processes. We refer to [5, 20, 25] for an overview. Recently the work of Lanz et al. [31, 32] brought some consolidation of expressing temporal constraints and defining the semantics of temporal constraints. The upper-bound constraints used here are correspond to TP 3 “maximum time lag between events”.

Early approaches checking temporal qualities of process definitions are [1, 19, 33] with techniques rooted in network analysis, scheduling, or constraint networks. These techniques stimulated the development of more advanced networks, the consideration of inter-organizational processes and for supporting temporal service level agreements for service compositions [4, 24].

Our approach is based on the algorithms for computing schedules for time constrained workflows presented in [17], but is extended to capture the uncertainty of the duration of contingent activities, and delivers a new formalization and proofs for soundness and completeness of the algorithms. This paper is an extension of [14] now including partial unfold as well as a set of additional experiments in the evaluation. The terminology was changed from history-dependent controllability to conditional controllability, as the latter, although introduced in the literature earlier (e.g. [10]) lead

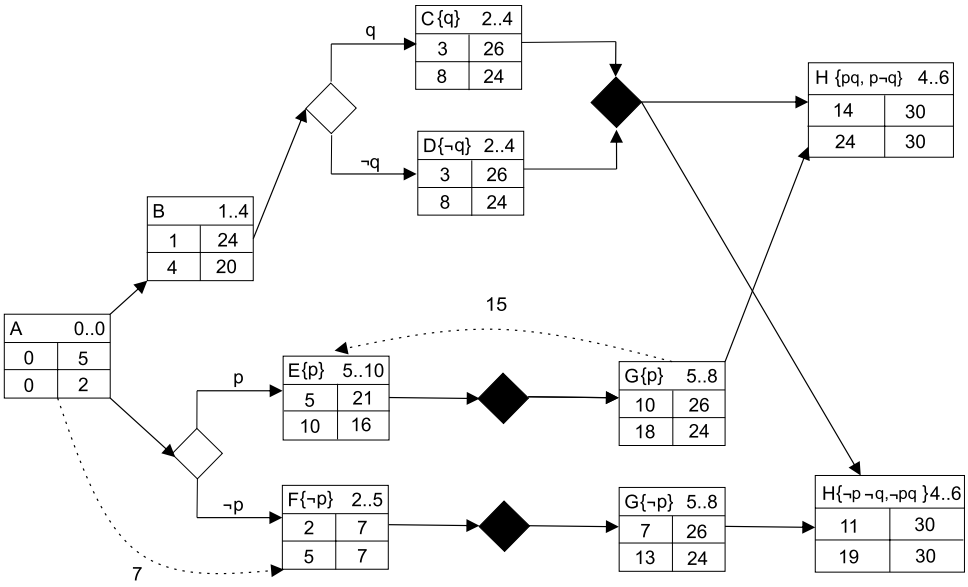

Fig. 6 Partially unfolded process graph with correct schedule (max 15xors)

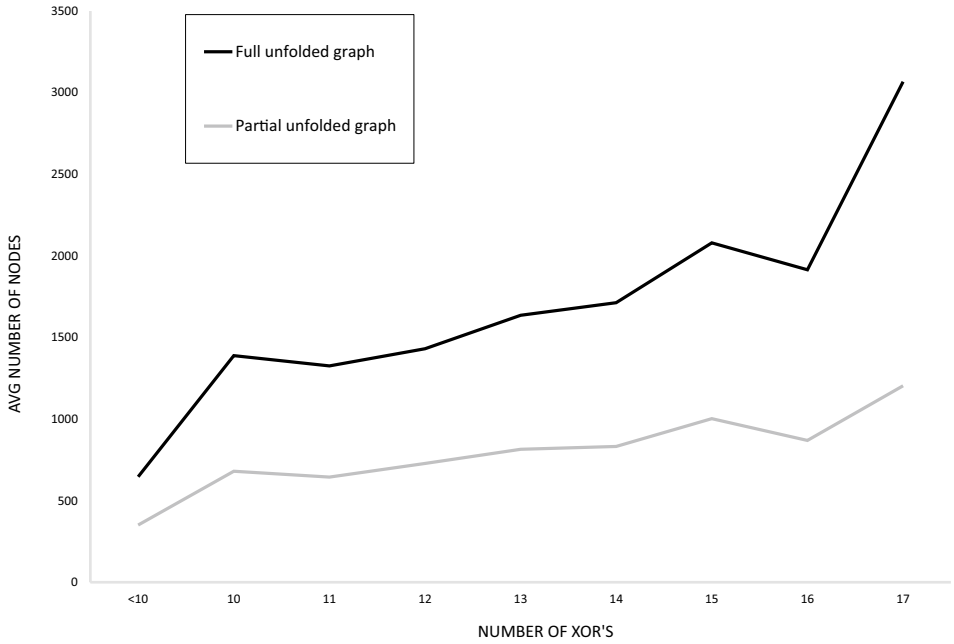

Fig. 7 Average number of nodes per XOR

to frequent misunderstanding with schedules derived from process logs (or process histories).

Another stream of research studies temporal constraint networks [12], which were applied for workflows in [1], and now reached the necessary expressiveness for analyzing business process models: checking the controllability [7] and dynamic controllability of temporal networks, as discussed in [8]. In particular, [6, 26, 28] present algorithms for checking the controllability of conditional simple temporal networks with a sound-and-complete algorithm for conditional simple temporal networks with uncertainty (CSTNU), the type of network corresponding to the process models discussed here. While these approaches check, whether an execution strategy avoiding time failures exists,

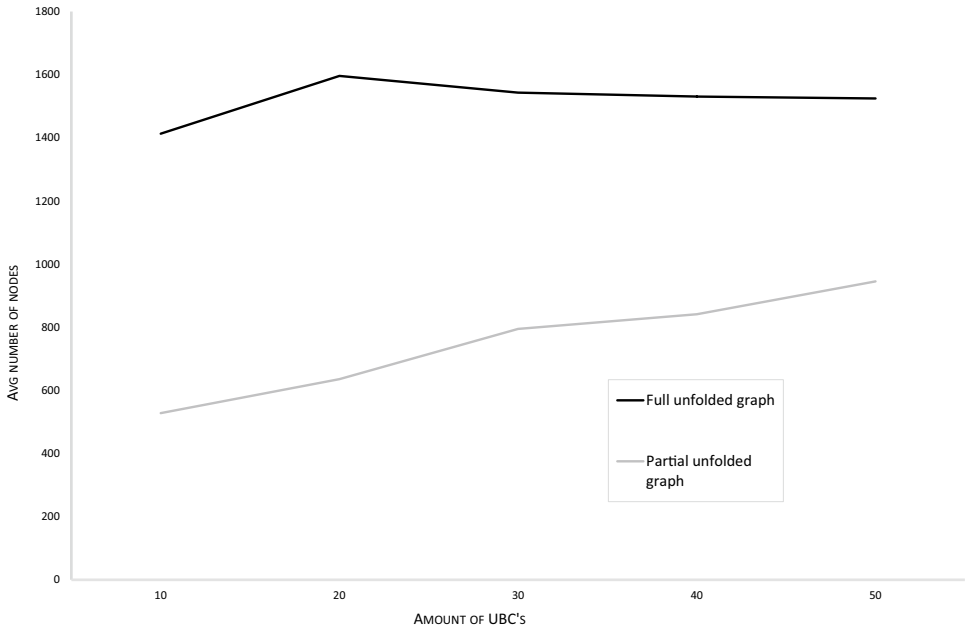

Fig. 8 Average number of nodes per upper-bound constraint

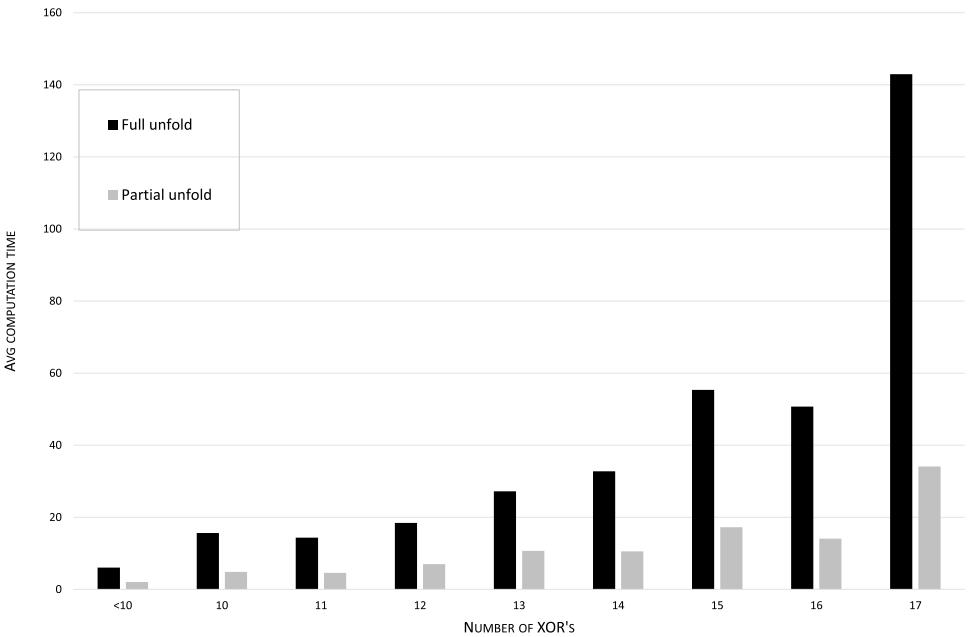

Fig. 9 Average time consumption per XOR in sec

our approach actually computes conditional schedules which guide the execution at run-time. For dynamic controllability, however, computing a schedule is not feasible.

Algorithms for other types of temporal constraint networks have been proposed in [2, 37]. Algorithms for computing conditional schedules for these types of networks is subject of ongoing research.

Conclusions

Among the most important quality criteria for the definition and execution of processes is to fulfill temporal requirements and do not violate temporal constraints. Time failures such as missed deadlines cause exception handling and might be costly. Therefore, we propose to use controllability and in particular conditional controllability as important characteristics for the quality of process definitions.

We presented a sound and complete algorithm for checking conditional controllability of temporally constrained process definitions, and for computing conditional schedules at design time. This algorithm provides a good basis for developing more efficient algorithms for the computation of conditional schedules with better average case complexity. The worst case complexity was shown to be exponential in the number of XOR-splits in a process. However, with the strategy to only partially unfold a process graph depending on the location of upper-bound constraints we achieved a significant improvement of both the size of the resulting graphs and the time for calculating conditional schedules. Our experiments showed that the proposed procedure for checking conditional controllability is feasible for realistic sized process models.

Our major aim is to compute schedules, and to support the time aware execution of processes avoiding temporal failures. To improve the qualities of schedules, in particular less rigid schedules with start time intervals and a good distribution of slack time as well as efficient algorithms for computing good schedules remain on our research agenda. Furthermore, efficient run-time monitoring of adherence to schedules and efficient computing of task deadlines when tasks are dispatched to actors is subject of ongoing research.

Acknowledgements Open access funding provided by University of Klagenfurt.

References

1. Bettini C, Wang XS, Jajodia S. Temporal reasoning in workflow systems. Distrib Parallel Databases. 2002;11(3):269–306.
2. Cairo M, Rizzi R. Dynamic controllability made simple. In: LIPIcs-Leibniz international proceedings in informatics, vol 90. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik; 2017.
3. Cairo M, Rizzi R. Dynamic controllability of simple temporal networks with uncertainty: simple rules and fast real-time execution. Theor Comput Sci. 2019;797:2–16.
4. Cardoso J, Sheth A, Miller J, Arnold J, Kochut K. Quality of service for workflows and web service processes. J Web Semant. 2004;1(3):281–308.
5. Cheikhrouhou S, Kallel S, Guermouche N, Jmaiel M. The temporal perspective in business process modeling: a survey and research challenges. Serv Oriented Comput Appl. 2015;9(1):75–85.
6. Cimatti A, Hunsberger L, Micheli A, Posenato R, Roveri M. Dynamic controllability via timed game automata. Acta Inform. 2016;53:681–722.
7. Combi C, Gozzi M, Posenato R, Pozzi G. Conceptual modeling of flexible temporal workflows. ACM Trans Auton Adapt Syst (TAAS). 2012;7(2):1–29.
8. Combi C, Hunsberger L, Posenato R. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty—revisited. In: Agents and artificial intelligence. Springer; 2014.
9. Combi C, Posenato R. Controllability in temporal conceptual workflow schemata. In: Business process management. Springer; 2009.
10. Combi C, Posenato R. Towards temporal controllabilities for workflow schemata. In: 2010 17th international symposium on temporal representation and reasoning. IEEE. 2010. p. 129–136.
11. Daniel F, Pernici B. Insights into web service orchestration and choreography. Int J E-Bus Res (IJEBR). 2006;2(1):58–77.
12. Dechter R, Meiri I, Pearl J. Temporal constraint networks. Artif Intell. 1991;49(1–3):61–95.
13. Dumas M, La Rosa M, Mendling J, Reijers HA, et al. Fundamentals of business process management, vol. 1. Berlin: Springer; 2013.
14. Eder J. Computing history-dependent schedules for processes with temporal constraints. In: Dang TK, üng JK, Takizawa M, Bui SH, editors, Future data and security engineering—6th international conference, FDSE 2019, Nha Trang City, November 27–29, 2019, Proceedings, Lecture Notes in Computer Science, vol 11814. Springer. 2019. p. 145–164.
15. Eder J, Franceschetti M, Köpke J. Controllability of business processes with temporal variables. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing. ACM. 2019. p. 40–47.
16. Eder J, Franceschetti M, Köpke J, Oberrauner A. Expressiveness of temporal constraints for process models. In: International conference on conceptual modeling. Springer. 2018. p. 119–133.
17. Eder J, Gruber W, Panagos E. Temporal modeling of workflows with conditional execution paths. In: Database and expert systems applications. Springer. 2000.
18. Eder J, Gruber W, Pichler H. Transforming workflow graphs. In: Interoperability of enterprise software and applications. London: Springer; 2006. p. 203–14.
19. Eder J, Panagos E, Rabinovich M. Time constraints in workflow systems. In: Advanced information systems engineering. Springer. 1999.
20. Eder J, Panagos E, Rabinovich M. Workflow time management revisited. In: Seminal contributions to information systems engineering. Springer. 2013.
21. Eder J, Pichler H. Response time histograms for composite web services. In: Proceedings. IEEE international conference on web services. IEEE. 2004. p. 832–33.

22. Esteves S, Veiga L. Waas: workflow-as-a-service for the cloud with scheduling of continuous and data-intensive workflows. Comput J. 2016;59(3):371–83.
23. Georgakopoulos D, Hornick M, Sheth A. An overview of workflow management: from process modeling to workflow automation infrastructure. Distrib Parallel Databases. 1995;3(2):119–53.
24. Guermouche N, Godart C. Timed model checking based approach for web services analysis. In: ICWS. IEEE. 2009. p. 213–21.
25. Hashmi M, Governatori G, Lam H, Wynn MT. Are we done with business process compliance: state of the art and challenges ahead. Knowl Inf Syst. 2018;57(1):79–133.
26. Hunsberger L, Posenato R. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In: IJCAI. 2018. p. 1324–30.
27. Hunsberger L, Posenato R, Combi C. The dynamic controllability of conditional STNs with uncertainty (2012). arXiv:1212.2005.
28. Hunsberger L, Posenato R, Combi C. A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In: Temporal representation and reasoning (TIME). IEEE. 2015.
29. Jajodia S, Kerschberg L. Advanced transaction models and architectures. Berlin: Springer Science & Business Media; 2012.
30. Lanz A, Posenato R, Combi C, Reichert M. Controllability of time-aware processes at run time. In: On the move to meaningful internet systems: OTM 2013 conferences. Springer. 2013.
31. Lanz A, Reichert M, Weber B. Process time patterns: a formal foundation. Inf Syst. 2016;57:38–68.
32. Lanz A, Weber B, Reichert M. Workflow time patterns for process-aware information systems. In: Enterprise: business-process and information systems modeling. Springer. 2010.
33. Marjanovic O, Orlowska M. On modeling and verification of temporal constraints in production workflows. Knowl Inf Syst. 1999;1(2):157–92.
34. Van Der Aalst WM. Workflow verification: Finding control-flow errors using petri-net-based techniques. In: Business process management: Springer. 2000.
35. Van Der Aalst WM, Ter Hofstede AH, Kiepuszewski B, Barros AP. Workflow patterns. Distrib Parallel Databases. 2003;14(1):5–51.
36. Vidal T, Fargier H. Contingent durations in temporal csps: from consistency to controllabilities. In: 4th international workshop on temporal representation and reasoning, TIME '97, Daytona Beach, May 10–11, 1997. IEEE Computer Society. p. 78–85.
37. Zavatteri M, Viganò L. Conditional simple temporal networks with uncertainty and decisions. Theor Comput Sci. 2019;797:77–101.