Advanced Temporal Control Structures for Business Process Modeling

Marco Franceschetti
Institute of Computer Science, University of St.Gallen, St.Gallen, Switzerland
marco.franceschetti@unisg.ch

Johann Eder
Department of Informatics-Systems, University of Klagenfurt, Klagenfurt am Wörthersee, Austria
johann.eder@aau.at

Abstract

Modeling time-constrained business processes is required in many domains but often challenging due to the difficulty of incorporating declarative temporal requirements in procedural process definitions. Here, we propose advanced temporal control structures based on highly expressive temporal conditions to give designers explicit control over the temporal behavior of a business process abstracting from low-level management of time aspects. These structures facilitate controlling executions to meet temporal requirements. We propose an approach for checking the temporal correctness of business processes featuring these structures in terms of dynamic controllability. The approach is based on the reduction to basic control structures and a further mapping to the CSTNUD, an expressive family of temporal constraint networks with established checking procedures.

Keywords: *Business Processes, Temporal Control Structures, Temporal Constraint Networks.*

1. Introduction

Modeling time-constrained business processes has recently gained increasing research interest. There are many approaches enriching process modeling languages, e.g., BPMN, with temporal constraints such as durations, deadlines, minimum and maximum allowed gaps between events (for overviews see, e.g, (Cheikhrouhou et al., 2015; Combi et al., 2019; Eder and Franceschetti, 2020; Hashmi et al., 2018; Pereira and Varajao, 2017)). These approaches differ in the control structures offered, in the expressiveness of temporal constraints (Lanz et al., 2014), in the formal model foundation, and in the underlying correctness models.

Workflow designers need tools to ensure that workflow controllers can steer the execution of processes without violating temporal constraints. Workflow designers face the challenges of *translating temporal requirements* into formal models (cf. (Cheikhrouhou et al., 2013; Chhetri et al., 2015; Franceschetti and Eder, 2021)), and of *checking temporal correctness* of the models, i.e., that they can be executed satisfying all temporal constraints (cf. (Chesani et al., 2018; Daniel and Pernici, 2006; Sheng et al., 2014; Watahiki et al., 2011)). Methods for checking time-constrained models against different notions of temporal correctness at design time have been successfully developed and deployed (Chesani et al., 2018; Eder et al., 2000; Lanz et al., 2013; Posenato, 2022; Watahiki et al., 2011).

Major temporal correctness criteria are satisfiability (or consistency), controllability, and dynamic controllability (cf. (Combi and Posenato, 2010)). Satisfiability or consistency checks for absence of contradictions, as studied in, e.g., (Cheikhrouhou et al., 2015; Chesani et al., 2018; Watahiki et al., 2011). For consistent models a valid execution exists; however, the controller might not be able to avoid time failures in all circumstances. Controllability requires that a schedule exists which satisfies all constraints (e.g., (Combi et al., 2012; Eder et al., 1999)). Dynamic controllability was established as the gold standard for temporal correctness of such models. It requires a correct dynamic execution strategy in which later time points only depend on earlier time points (but not vice versa). It admits that a controller has the ability to satisfy all temporal constraints despite uncertainty (e.g., xor-splits, uncertain activity durations) under any foreseeable circumstance.

Rather recently, some approaches considering temporal conditions in process control constructs like xor-splits or while-loops were proposed (e.g., (Pichler et al., 2017)). They provide designers with the possibility to explicitly reflect the temporal situation in process models and react to it. The general operational semantics of these constructs is straightforward: when

a temporal control structure is executed its temporal condition is evaluated, emitting a Boolean value that decides the continuation of the process. The formal treatment of these temporal control structures and checking the temporal correctness of process models with these structures turned out to be difficult. Some approaches do not take the idiosyncrasies of temporal conditions into account and treat them like any other conditions as *oracles*, i.e. their result cannot be influenced and is independent of any other aspect.

However, designer might relate the condition to other temporal properties of a process and thus know the result of the condition evaluation depends on the temporal situation at runtime (e.g. on the path executed before). Moreover, Temporal xor-splits have another interesting property: while xor conditions are dealt with as oracles (i.e., their results can only be observed but not influenced), controllers are sometimes able to influence the result of evaluating a temporal condition by delaying its evaluation.

Moreover, existing approaches for checking dynamic controllability are only applicable to process models that are acyclic or only contain loops with a limited number of iterations. If a process contains a loop whose number of iterations is only bound by a Boolean condition (e.g., a while loop), it cannot be dynamically controllable, although it can be checked for satisfiability, as shown in, e.g., (Chesani et al., 2018).

To the best of our knowledge, this combination of features (temporal conditions not treated as oracles, checking dynamic controllability, allowing loops with indefinite number of iterations) so far has only be studied in (Eder et al., 2018). They showed how the Conditional Simple Temporal Network with Uncertainty and Decisions (CSTNUD) (Zavatteri, 2017), a highly expressive temporal constraint network (cf. (Dechter et al., 1991)), can be used to check temporal correctness of business processes featuring temporal xor. Further work in (Franceschetti and Eder, 2019) showed that the temporal loop can be transformed into a temporal xor. However, these approaches only permitted very basic temporal conditions (time elapsed since process start).

The goal of this paper is to present a full-fledged proposal for *advanced* temporal control structures with complex conditions referring to *any* prior event in a process model, composed by conjunctions and disjunctions, together with a procedure for checking their dynamic controllability. A further goal is to break a quite painful restriction of time-constrained processes: that the formalism of temporal constraint networks only support acyclic process models – they do not support models with *while* loops, as such loops can never be dynamically controllable. With t-loops we finally offer a loop construct manageable by temporal constraint networks for checking dynamic controllability.

This paper is structured as follows: Section 2 provides a motivating example and recalls relevant concepts; Sect. 3 presents a compact process model including the advanced temporal control structures; Sect. 4 illustrates transformations of the advanced temporal control structures to enable checking dynamic controllability; Sect. 5 concludes the paper.

2. Motivation and Background

2.1. Motivating Example

Fig. 1 shows, in BPMN-like notation (cf. Allweyer, 2016), a time-constrained process that includes the t-split (in the example we do not focus on the t-loop, since the t-split is the fundamental element at the basis of the t-loop, as we will show later). Activity duration ranges are specified as an interval after their labels. The process starts with the arrival of a new order. First, confirmation is sent to the customer (*C* in the figure), then the order is processed (*P* in the figure). (Manual) activity *G* consists in gathering the ordered goods from various warehouses distributed across several locations, and preparing the goods for shipping. This task can take some time between 8 and 72 hours. Then, some activities are executed in parallel: *I* for sending an invoice to the customer, followed by *A* for archiving the order information; at the same time, the ordered items are shipped. The vendor guarantees delivery between 24 and 96 hours after the order is confirmed (temporal constraints in Fig. 1, bottom left). If less than 24 hours are elapsed since order confirmation or less than 16 hours are elapsed since goods gathering started, standard shipment (*S*, taking 24 to 72 hours) is used; otherwise express courier (*E*, taking 12 to 24 hours) (*cond* in Fig. 1). Quality management imposes that the process terminates at most 168 hours after completion of *P* (temporal constraint in Fig. 1, top). If activity durations cannot be controlled, the vendor can check dynamic controllability of the process to know whether all the temporal constraints can be met under any circumstance, fulfilling its contractual obligations despite uncertainty.

2.2. Basic Temporal Control Structures

The temporal xor and temporal loop introduced in (Pichler et al., 2017), hereinafter basic t-split and basic t-loop, have a simple temporal condition, modeling the requirement that the process continuation depends on the time elapsed at their execution time since the process started. At run time, this elapsed time is determined by the duration of previously executed activities, observed

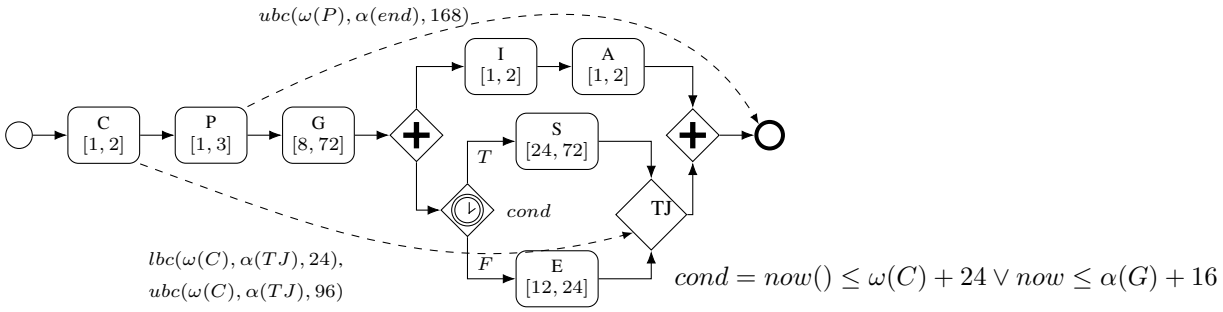


Figure 1. Example with temporal gateway and temporal constraints. $\alpha(X), \omega(X)$ denote the start, resp. end events of flow element X. Temporal constraints are drawn as dashed arrows, with ubc signifying an upper-bound and lbc a lower-bound constraint. For simplicity, gateways have implicit duration $[0, 0]$.

conditions, and the potential delay enforced before starting executing the basic t-split, resp. t-loop.

In Fig. 1, a gateway with a clock icon models the choice whether to use standard shipping or express courier. However, the basic t-split as defined in (Pichler et al., 2017) is not capable of encoding the condition specified in the example. In (Pichler et al., 2017), the temporal condition is formally defined as $\eta \leq c$, where η is the elapsed time since the start of the process, and c is some constant value specified by the designer. The basic t-split condition requires that the execution continues with the t-split "$true$" branch if no more than c time units are elapsed since process start, with the "$false$" branch otherwise. Such a condition is, however, too restrictive for many applications that may require more advanced temporal conditions to steer the execution, for instance referring to events other than the time of execution start or combining several conditions at once.

In this paper, we extend the condition of the basic t-split and basic t-loop to enable higher expressiveness. This proposed condition is a conservative extension of the simple one introduced in (Pichler et al., 2017).

2.3. Temporal Correctness

Checking temporal correctness has been extensively studied on Temporal Constraint Networks (TCNs). A systematic analysis of all TCN types is beyond the scope of this paper: here, we focus on the Conditional Simple Temporal Network with Uncertainty and Decisions (CSTNUD) (Zavatteri, 2017), used to check process temporal correctness in, e.g., (Combi et al., 2019). The CSTNUD models temporal problems in which there exist contingent time points and observed conditions, i.e., whose value is not assigned by the executor but only observed at run time, and decision points. A decision point allows the controller to decide on the continuation of the CSTNUD execution by setting the value of some propositional letter representing the decision. This is in contrast to observation points, which steer the execution depending on some observed uncontrollable condition.

A CSTNUD is dynamically controllable (DC) if it admits a *dynamic execution strategy* assigning values to the controllable time points and decisions in response to the values the uncontrollable time points and conditions are observed to take during execution, so that all temporal constraints are satisfied. Effective procedures for checking whether a CSTNUD is DC have been proposed in (Zavatteri, 2017; Zavatteri et al., 2022).

3. Advanced Temporal Control Structures

3.1. Process Model

In line with (Franceschetti and Eder, 2019) and to support checking temporal correctness using TCNs, we formally define a process model supporting the most common time and control flow patterns (Lanz et al., 2014; van Der Aalst et al., 2003). Control flow elements have a start and an end event. Events take values from the set $\mathbb{N}_0$, i.e., an event coincides with a point on an increasing discrete time axis, and represents the temporal distance to a reference time point *zero*, assumed to be the start of the process. The actual distance between the values for a start and its corresponding end event is the actual duration of a control flow element instance. Here, we assume without loss of generality that all activities have a contingent duration, i.e., it can only be observed, but not controlled.

Definition 1 (Process Model) *A process P is a tuple (N, E, T, L, C) where:*

- *N is a set of nodes, from which we derive the set*

of control flow events $N^e = \{\alpha(n), \omega(n)\}$, where $\alpha(n)$ and $\omega(n)$ are the start and end events of n.

- *$E \subseteq N \times N$ is a set of directed edges, defining precedence constraints. The predecessors and successors of a node n are $n.Pred = \{m|(m,n) \in E\}$ and $n.Succ = \{m|(n,m) \in E\}$. $n.Pred^+, n.Succ^+$ are the respective transitive closures.*
- *(N, E) forms an acyclic graph.*
- *$T : N \rightarrow$ {activity, start, end, xor-split, t-split, d-split, merge, and-split, and-join, t-loop, block} is a function associating each node with a type.*
- *$L : E \rightarrow \{\text{“}T\text{”}, \text{“}F\text{”}\}$ is a labeling function: $\forall n \in N$: $T(n) \in$ {xor-split, d-split, t-split, t-loop}, $\exists (n, m_1), (n, m_2) \in E$, with $m_1 \neq m_2$: $L((n, m_1)) = \text{“}T\text{”}$, $L((n, m_2)) = \text{“}F\text{”}$.*
- *C is a set of temporal constraints consisting of*
  - *duration constraints $\forall n \in N$: $dur(n, n.d_{min}, n.d_{max})$, defining the allowed duration span for executing a node;*
  - *upper-bound constraints: $ubc(a, b, \delta)$, where $a, b \in N^e, \delta \in \mathbb{N}_0$, defining a maximum allowed timespan δ between events a and b;*
  - *lower-bound constraints: $lbc(a, b, \delta)$, where $a, b \in N^e, \delta \in \mathbb{N}_0$, defining a minimum allowed timespan δ between events a and b.*

At run time, for an and-split node, all outgoing flows are executed; for a xor-split node, only one outgoing flow is executed based on an observed condition; for a d-split node, only one flow is executed based on the controller decision. For compatibility with (Zavatteri, 2017), in the following we assume that a process model is block-structured (Dumas et al., 2012), with proper nesting of matching pairs of and/xor/t/d gateways. We assume that a process has exactly one start node with no predecessor, and one end node with no successor. So the number of predecessors $|n.Pred|$ per node type is as follows: 0 for start, 2 for merge, 2 for and-join, and 1 for all other types. The number of successors $|n.Succ|$ per node-type is as follows: 0 for end, 2 for xor-split, t-split, t-loop, d-split, and-split, and 1 for all other types.

In general, each branch of a t-split may have an arbitrary number of nodes of arbitrary type. For ease of formalization and without loss of generality we assume that all t-splits have exactly one node in the “$true$” and one node in the “$false$” branch, each of type $block$, where a node of type $block$ is a sub-process encapsulation. Similarly, we assume that also all t-loop nodes are followed by exactly one node of type $block$. Additionally, we forbid temporal constraints crossing sub-process boundaries since they refer to events that are not visible outside the sub-process scope and require special treatment similar to choreographies (Franceschetti and Eder, 2021).

Each t-split and t-loop node n has a condition $n.cond$, which is a clause recursively defined as follows:

Definition 2 (Clause) *A clause c for the condition of a t-split/t-loop node n is either:*

- *disjunctive, if $c = c_1 \vee c_2$, with c_1, c_2 clauses, or*
- *conjunctive, if $c = c_1 \wedge c_2$, with c_1, c_2 clauses, or*
- *atomic, if $c = now() \leq e + \delta$, with $e \in N^e$, $\delta \in \mathbb{N}_0$.*

It is easy to see that the condition of a t-split/t-loop, in the general case, is a set of atomic clauses stitched together by conjunctions and disjunctions (thus, we denote with $c \in n.cond$ the fact that an atomic clause c is part of the condition $n.cond$ of a node n). These atomic clauses may refer to different control flow events. For the condition to be evaluated, it is necessary that these events occur before the execution of the t-split/t-loop. Therefore, we introduce the definition of well-formedness to formalize such a requirement:

Definition 3 (Well-formedness) *Let $P(N, E, T, L, C)$ be a process. Let n be a node in N with $T(n) \in \{t\text{-}split, t\text{-}loop\}$. We say that n is well-formed if $\forall c = now() \leq e + \delta \in n.cond$, $\exists m \in n.Pred^+ : e = \alpha(m) \vee e = \omega(m)$.*

Below, we consider only well-formed t-splits and t-loops. The definition of well-formedness forbids conditions that refer to events of process nodes occurring in parallel. This prevents awkward run time behaviors in which the controller is forced to delay a t-split/t-loop execution in order for the events referenced in the condition to take place. Indeed, we assume that at the execution of a t-split/t-loop its condition is evaluated immediately with no delays. However, it should be in the decisional power of the controller to determine when a node is executed. Consider the t-split from the motivating example with the disjunctive condition $now() \leq \omega(C) + 24 \vee now() \leq \alpha(G) + 16$. Assuming that $\omega(C) = 1$, $\alpha(G) = 10$, if the t-split is executed before or at time point 26 then the “$true$” path is chosen; if it is executed after time point 26 the “$false$” path is selected. The process controller might decide to delay the execution of the t-split after time point 26, thus controlling the result of the condition evaluation, and deciding to follow the “$false$” path.

3.2. Temporal Semantics

We define the temporal semantics of the process model in terms of scenarios. A scenario $\bar{S}(P)$ of a process model $P(N, E, T, L, C)$ is a connected sub-graph of P containing the start and the end node, in which each xor-split, d-split, and t-split has exactly 1 successor, and all start and end events are associated with a value representing their execution time.

Definition 4 (Scenario) *Let $P(N, E, T, L, C)$ be a process. Let W be a function assigning real values to events. $\bar{S}(P) = (N', E', W)$ is a scenario for P, iff*

1. $N' \subseteq N,\ E' \subseteq E$.
2. $\forall n \in N,\ T(n) = start \rightarrow n \in N'$.
3. $\forall n \in N,\ \forall (n, m) \in E$:
   - $T(n) \notin \{xor\text{-}split, d\text{-}split, t\text{-}split\} : n \in N' \rightarrow m \in N', (n, m) \in E'$.
   - $T(n) \in \{xor\text{-}split, d\text{-}split, t\text{-}split\} : (n, m_1) \in E, (n, m_2) \in E, m_1 \neq m_2, n \in N' \rightarrow (n, m_1) \in E' \leftrightarrow (n, m_2) \notin E'$.
4. $\forall n \in N',\ W(\alpha(n)) = \tau_s,\ W(\omega(n)) = \tau_e$, *with* $\tau_s, \tau_e \in \mathbb{N}_0$.

A scenario is valid if it satisfies all temporal constraints; for nodes followed by a conditional execution, we require that the proper successor is included in the scenario: if a t-split condition evaluates to *False*, then the "$false$" successor is included in the scenario, otherwise the "$true$" successor.

Definition 5 (Valid Scenario) *Let $P(N, E, T, L, C)$ be a process. Let $\bar{S}(P) = (N', E', W)$ be a scenario for P. $\bar{S}(P)$ is valid, iff:*

1. $\forall n \in N', W(\alpha(n)) \leq W(\omega(n))$.
2. $\forall n \in N'$: $dur(n, d_{min}, d_{max}) \in C,\ W(\alpha(n)) + n.d_{min} \leq W(\omega(n)) \leq W(\alpha(n)) + n.d_{max}$.
3. $\forall (m, n) \in E',\ W(\omega(m)) \leq W(\alpha(n))$.
4. $\forall (n, m) \in E' : T(n) \in \{xor\text{-}split, t\text{-}split, d\text{-}split, t\text{-}loop\}$:
   - $n.cond \models True \rightarrow L((n, m)) =$ "T".
   - $n.cond \models False \rightarrow L((n, m)) =$ "F".
5. $\forall ubc(a, b, \delta)\ a, b \in N'^e, W(b) \leq W(a) + \delta$.
6. $\forall lbc(a, b, \delta)\ a, b \in N'^e : W(b) > W(a) + \delta$.

A valid scenario formally defines the admissible values for events satisfying the temporal constraints in a process, thus defining the process temporal semantics.

4. Checking Dynamic Controllability

To check the dynamic controllability of processes with t-splits and t-loops, we follow a transformation approach, mapping process models with t-splits and t-loops to process models without t-splits and t-loops.

We transform each temporal control structure with a complex (non-atomic) condition, by unfolding it in a nesting of t-splits, resp. t-loops with atomic conditions. A new t-split, resp. t-loop is inserted either in the "$true$" or in the "$false$" branch of the original t-split, resp. t-loop, depending on whether the condition was a conjunction or a disjunction. These transformations are applied until all temporal control structures have atomic conditions. Then, we transform any t-loop into a t-split with the same dynamic controllability, but not execution, semantics. Finally, we transform each t-split with atomic condition to a decision node, and express the t-split condition through upper- and lower-bound constraints on the successors of the decision node.

We can apply these transformations for all temporal control structures in the original process model and derive a process model without any temporal control structures. This resulting process model conforms with the process model used in (Zavatteri, 2017), where it is then further transformed into a CSTNUD for checking dynamic controllability. Recall that we consider process models with only well-formed t-splits and t-loops.

4.1. Transformation of t-split Conditions

To transform a t-split n with a condition $n.cond$ that is not an atomic clause, we define one-step unfolding transformations. Such transformations extract one conjunct, resp. disjunct from $n.cond$, and introduce a new t-split in the "$true$", resp. "$false$" branch of n with the extracted conjunct, resp. disjunct as its condition. The iterated step application generates a nested t-splits structure, each with an atomic condition. Recall that, with no loss of generality, we assume that a t-split has exactly one node in the "$true$" branch and one node in the "$false$" branch, both of type $block$.

Disjunctive clause: If the t-split condition is of the form $c1 \vee c2$, we transform it to an equivalent nested structure of two t-splits, each with one of the two disjuncts as its condition, where the second one is nested into the "$false$" branch of the first one (see Fig. 2). To enforce the evaluation of both conditions at the same time as it would be in the original t-split, an upper- and a lower-bound constraint, both with $\delta = 0$, are placed between the two t-splits.

Definition 6 (ts-unfold-or) *Let $P(N, E, T, L, C)$ be a process model. Let $n, m_1, m_2, nj \in N$, with $T(n) =$*

t-split, $T(nj) = t\text{-}join$*,* $n.cond = c1 \vee c2$*, and with* $dur(m_1, m_{1min}, m_{1max}) \in C$*. Let* $(n, m_1), (n, m_2) \in E$*, with* $L(n, m_1) =$ "T"*,* $L(n, m_2) =$ "F"*.*

Then the ts-unfold-or transformation is defined as $\mathcal{T}^S_\vee(P, n) = P'(N', E', T', L', C')$*, where:*

- $N' = N \cup \{n', m'_1, nj'\}$, $n.cond = c1$, $n'.cond = c2$.
- $E' = E \setminus \{(n, m_2)\} \cup \{(n, n'), (n', m'_1), (m'_1, nj'), (n', m_2), (m_2, nj'), (nj', nj)\}$.
- $T' = T \cup \{T(n') = t\text{-}split,\ T(m'_1) = T(m_1), T(nj') = t\text{-}join\}$.
- $L' = L \setminus \{L(n, m_1), L(n, m_2)\} \cup \{L(n, n') =$ "F", $L(n', m'_1) =$ "T", $L(n', m_2) =$ "F"$\}$.
- $C' = C \cup \{lbc(\alpha(n), \alpha(n'), 0), ubc(\alpha(n), \alpha(n'), 0), dur(n', 0, 0), dur(m'_1, m_{1min}, m_{1max})\}$.

Conjunctive clause: If the t-split condition is of the form $c1 \wedge c2$, we transform it to an equivalent nested structure of two t-splits, each with one of the two disjuncts as its condition, where the second one is nested into the "$true$" branch of the first one (see Fig. 3). To enforce the evaluation of both conditions at the same time as it would be in the original t-split, a pair of upper- and lower-bound constraints with $\delta = 0$ is inserted between the two t-splits.

Definition 7 (ts-unfold-and) *Let* $P(N, E, T, L, C)$ *be a process model. Let* $n, m_1, m_2, nj \in N$*, with* $T(n) = t\text{-}split$*,* $T(nj) = t\text{-}join$*,* $n.cond = c1 \wedge c2$*, and with* $dur(m_2, m_{2min}, m_{2max}) \in C$*. Let* $(n, m_1), (n, m_2) \in E$*, with* $L(n, m_1) =$ "T"*,* $L(n, m_2) =$ "F"*.*

Then the ts-unfold-and transformation is defined as $\mathcal{T}^S_\wedge(P, n) = P'(N', E', T', L', C')$*, where:*

- $N' = N \cup \{n', m'_2, nj'\}$, $n.cond = c1$, $n'.cond = c2$.
- $E' = E \setminus \{(n, m_1)\} \cup \{(n, n'), (n', m_1), (m_1, nj'), (n', m'_2), (m'_2, nj'), (nj', nj)\}$.
- $T' = T \cup \{T(n') = t\text{-}split,\ T(m'_1) = T(m_1), T(nj') = t\text{-}join\}$.
- $L' = L \setminus \{L(n, m_1), L(n, m_2)\} \cup \{L(n, n') =$ "T", $L(n', m_1) =$ "T", $L(n', m_2) =$ "F"$\}$.
- $C' = C \cup \{lbc(\alpha(n), \alpha(n'), 0), ubc(\alpha(n), \alpha(n'), 0), dur(n', 0, 0), dur(m'_2, m_{2min}, m_{2max})\}$.

We now show the soundness of transformations *ts-unfold-or* and *ts-unfold-and* from Definitions 6, 7.

Lemma 1 *The application of transformation* ts-unfold-or*, resp.* ts-unfold-and *from Def. 6, resp. Def. 7 to a t-split node* n *in a process model* P *results in a process model* P'*:*

1. *that only has well-formed t-splits,*
2. *whose scenarios are equivalent to the scenarios of the original process model.*

Proof 1 *We prove the two aspects separately:*

1. *For well-formedness: with both transformations, a new t-split node* n' *is introduced as a successor of the original t-split node* n *with the precedence constraint* $(n, n') \in E'$*. The pair of constraints* $\{lbc(\alpha(n), \alpha(n'), 0), ubc(\alpha(n), \alpha(n'), 0)$ *enforce the occurrence of* n' *to be at the same time as the occurrence of* n*. Since* n *is a well-formed t-split node, in the original process model all events referenced in* $c1 \vee c2$*, resp.* $c1 \wedge c2$ *occur before* n*. Then, in the model resulting from the application of* ts-unfold-or*, resp.* ts-unfold-and*, all events referenced in* $c1$ *and in* $c2$ *still occur before* n *and* n'*. From this, we conclude that* n *and* n' *are both well-formed in* P' *according to Definition 3.*

2. *For the equivalence of the scenarios: for each of the transformations, any valid scenario for the original process model subsumes a valid scenario for the derived process model, and any valid scenario for the derived process model subsumes a valid scenario for the original process model. We show this fact for transformation* ts-unfold-and*, omitting the case of transformation* ts-unfold-or *since it is similar.*
   Let $\overline{S}(P) = (N'_P, E'_P, W_P)$ *be a valid scenario for the original process model* P*. Let* P' *be the process model resulting from the application of ts-unfold-and to a node* n *in* P*. For all elements of* P' *that also belong to* P*,* $\overline{S}(P') = (N'_{P'}, E'_{P'}, W_{P'})$ *is defined the same as* $\overline{S}(P)$*, i.e., it satisfies conditions 1–6 from Definition 5. We show that we can extend function* $W_{P'}$ *for assigning values to the elements of* P' *not in* P *(i.e.,* n'*) that also satisfy conditions 1–6. We define* $W_{P'}(\alpha(n')) = W_{P'}(\omega(n')) = W_P(\alpha(n)) = W_P(\omega(n))$ *and* $W_{P'}(\alpha(m'_2)) = W_P(\alpha(m_2))$ *and* $W_{P'}(\omega(m'_2)) = W_P(\omega(m_2))$*. With this extension:*
   (a) *Condition 1 is trivially satisfied since* $W_{P'}(\alpha(n')) = W_{P'}(\omega(n'))$*.*

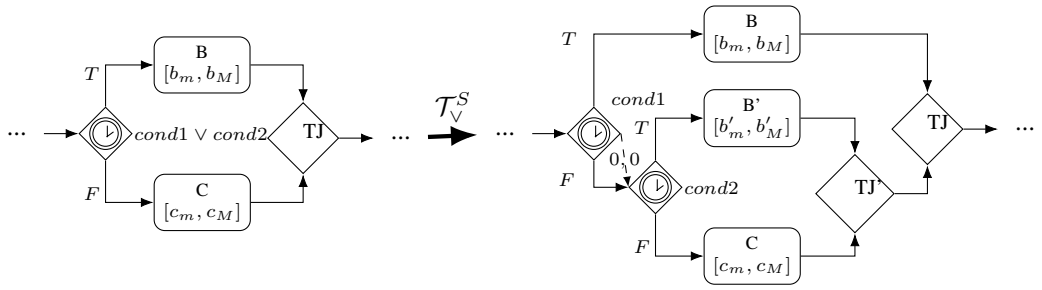


Figure 2. Unfolding of t-split with disjunctive clause condition.

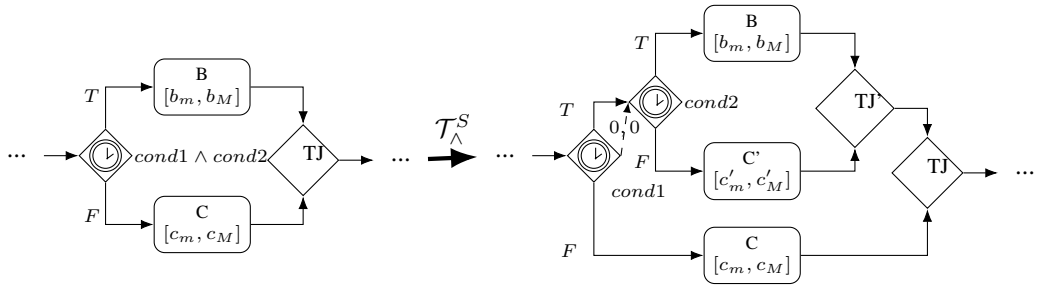


Figure 3. Unfolding of t-split with conjunctive clause condition.

(b) Condition 2 is satisfied since $W_{P'}(\alpha(n')) + 0 \leq W_{P'}(\omega(n')) \leq W_{P'}(\alpha(n')) + 0$ *and* $W_P(\alpha(m_2)) + m_2.d_{min} \leq W_P(\omega(m_2)) \leq W_P(\alpha(m_2)) + m_2.d_{max}$.

(c) Condition 3 is satisfied since $W_{P'}(\omega(n)) = W_P(\omega(n)) = W_{P'}(\alpha(n'))$.

(d) Condition 4 is satisfied by construction with the application of ts-unfold-and.

(e) Conditions 5–6 are satisfied since $W_{P'}(\alpha(n)) = W_{P'}(\alpha(n')) = W_{P'}(\omega(n'))$.

Therefore, $\overline{S}(P')$ *is a valid scenario for* P'.
Let $\overline{S}(P')$ *be a valid scenario for the derived process model* P'. *We can define* $\overline{S}(P) = \overline{S}(P')$ *for all elements of* P, *i.e., all elements but* n'. *For these process elements, conditions 1–6 from Definition 5 still hold in* $\overline{S}(P)$. *Therefore,* $\overline{S}(P)$ *is a valid scenario for* P.

4.2. Transformation of t-loop Conditions

The work in (Franceschetti and Eder, 2019) introduced a transformation of the basic t-loop that preserves its dynamic controllability semantics. Such a transformation enables checking the dynamic controllability of a process model with basic t-loops by reducing it to a process model without any t-loops. The assumption in (Franceschetti and Eder, 2019) is that the t-loop has a number of iterations of a t-loop is finite and limited by the temporal loop condition. The condition of the advanced t-loop we introduce here (i.e., *now* is not later than some fixed points in time) induces the same boundedness property (since time flows monotonically), allowing us to define a similar transformation. However, we first need to handle composite clauses that may define a t-loop condition.

Disjunctive clause: Similar to the t-split, if the t-split condition is of the form $c1 \vee c2$, we transform it to an equivalent nested structure of two t-loops, each with one of the two disjuncts as its condition, inserting the second one into the "$false$" branch of the first one.

Definition 8 (tl-unfold-or) *Let* $P(N, E, T, L, C)$ *be a process model. Let* $n, b, m \in N$, *with* $T(n) =$ *t-loop,* $T(b) = block$, $n.cond = c1 \vee c2$, *and with* $dur(b, b_{min}, b_{max}) \in C$. *Let* $(n, b), (n, m) \in E$ *with* $L(n, b) =$ "T", $(n, m) =$ "F".

Then the tl-unfold-or transformation is defined as $\mathcal{T}^L_\vee(P, n) = P'(N', E', T', L', C')$, *where:*

- $N' = N \cup \{n', b'\}$, $n.cond = c1$, $n'.cond = c2$.
- $E' = E \setminus \{(n, m)\} \cup \{(n, n'), (n', b'), (n', m), (b', n')\}$.
- $T' = T \cup \{T(n') =$ *t-loop,* $T(b') = block\}$.
- $L' = L \setminus \{L(n, m)\} \cup \{L(n, n') =$ "F", $L(n', b') =$ "T", $L(n', m) =$ "F"$\}$.
- $C' = C \cup \{lbc(\alpha(n), \alpha(n'), 0), ubc(\alpha(n), \alpha(n'), 0), dur(n', 0, 0), dur(b', b_{min}, b_{max})\}$.

Conjunctive clause: Similar to the t-loop, if the t-split condition is of the form $c1 \wedge c2$, we transform it to an equivalent nested structure of two t-splits, each with one of the two disjuncts as its condition, inserting second one into the "$true$" branch of the first one.

Definition 9 (tl-unfold-and) *Let $P(N, E, T, L, C)$ be a process model. Let $n, b, m \in N$, with $T(n) =$ t-loop, $T(b) =$ block, $n.cond = c1 \wedge c2$, and with $dur(b, b_{min}, b_{max}) \in C$. Let $(n, b), (n, m) \in E$ with $L(n, b) =$ "T", $(n, m) =$ "F".*

Then the tl-unfold-and transformation is defined as $\mathcal{T}^L_\wedge(P, n) = P'(N', E', T', L', C')$, where:

- $N' = N \cup \{n'\}$, $n.cond = c1$, $n'.cond = c2$.
- $E' = E \setminus \{(n, b), (b, n)\} \cup \{(n, n'), (n', b), (n', m), (b', n')\}$.
- $T' = T \cup \{T(n') = \text{t-loop}\}$.
- $L' = L \setminus \{L(n, b)\} \cup \{L(n, n') =$ "T", $L(n', b) =$ "T", $L(n', m) =$ "F"$\}$.
- $C' = C \cup \{lbc(\alpha(n), \alpha(n'), 0), ubc(\alpha(n), \alpha(n'), 0), dur(n', 0, 0)\}$.

We omit the proof of the soundness of transformations $\mathcal{T}^L_\vee$ and $\mathcal{T}^L_\wedge$ since Lemma 1 provided formal proof of the soundness of the corresponding transformations for the case of the t-split, and such a proof can be easily adapted to the case of the t-loop.

The unfolding transformation steps can be applied iteratively to a process model until there are only t-splits and t-loops with atomic clause conditions, which we call a full unfolding of the temporal control structures. In the following, we consider processes in which all t-splits and t-loops have been fully unfolded.

4.3. Transformation of t-loops into t-splits

Transformations $\mathcal{T}^L_\vee$ and $\mathcal{T}^L_\wedge$ can be applied to all t-loop nodes of a process model until all have atomic conditions. Then, we can apply the transformations in (Franceschetti and Eder, 2019) for the basic t-loop, and reduce the advanced t-loop to an advanced t-split with atomic condition (see Fig. 4). The resulting process model is obviously not execution equivalent to the original one; however, the two process models are dynamic controllability equivalent, i.e., the transformed model is dynamically controllable if and only if the original model is dynamically controllable.

Definition 10 (tl-transform) *Let $P(N, E, T, L, C)$ be a process model. Let $n, b, m \in N$, with $T(n) =$ t-loop, $T(b) =$ block, $n.cond = now \leq e + \delta$, and with $dur(b, b_{min}, b_{max}) \in C$. Let $(l, n), (n, b), (n, m) \in E$ with $L(n, b) =$ "T", $(n, m) =$ "F".*

Then the tl-transform transformation is defined as $\mathcal{T}^L_T(P, n) = P'(N', E', T', L', C')$, where:

- $N' = N \setminus \{n\} \cup \{ts, tj\}$, $ts.cond = n.cond$,
- $E' = E \setminus \{(l, n), (n, b), (b, n), (n, m)\} \cup \{(l, ts), (ts, b), (ts, tj), (b, tj), (tj, m)\}$,
- $T' = T \cup \{T(ts) = \text{t-split}, T(tj) = merge\}$,
- $L' = L \setminus \{L(l, n), L(n, b), L(b, n), L(n, m)\} \cup \{L(ts, b) =$ "T", $L(ts, tj) =$ "F"$\}$,
- $C' = C \setminus \{dur(b, b_{min}, b_{max})\} \cup \{lbc(e, b.s, \delta + b_{min}), dur(b, 0, b_{max} - b_{min}), dur(ts, 0, 0), dur(tj, 0, 0)\}$.

Transformation $\mathcal{T}^L_T$ replaces the t-loop with a t-split representing the latest possible iteration of the t-loop block. The introduced temporal constraints enforce the latest possible time for executing the t-loop block at most c time units after event e referenced in the original condition, and enforce duration bounds for the last t-loop block iteration. The t-split condition forces the t-loop block to be skipped only if its condition evaluates to "$False$". The valid scenarios for the original process model and the transformed process model are equivalent, hence $\mathcal{T}^L_T$ preserves the dynamic controllability. Applying $\mathcal{T}^L_T$ to all t-loops with atomic condition leads to an acyclic process model with t-splits. This transformation of a process model with (temporal) loops to an acyclic model is possible as we do not deal with the temporal conditions as oracles but are able to consider them in relation to other temporal constraints and are thus able to limit the unfolding of temporal loop to a finite acyclic structure.

4.4. Checking Dynamic Controllability

When t-splits have an atomic clause condition, they can be transformed into decision nodes (d-splits in the process model of Def. 1) (Eder et al., 2018). Basically, a t-split is replaced by a d-split with one additional dummy successor in the "$true$" branch; the condition of the t-split is translated to upper-bound and lower-bound constraints over the dummy node and the "$false$" successor of the split, respectively. We refer to (Eder et al., 2018) for a formalization of the transformation replacing a t-split node in a process with a d-split node, and the corresponding proof of equivalence. With these results, we can start with a process with t-split nodes having complex conditions, and eventually derive an equivalent process without t-split nodes. The difference with the approach presented in (Eder et al., 2018) is that

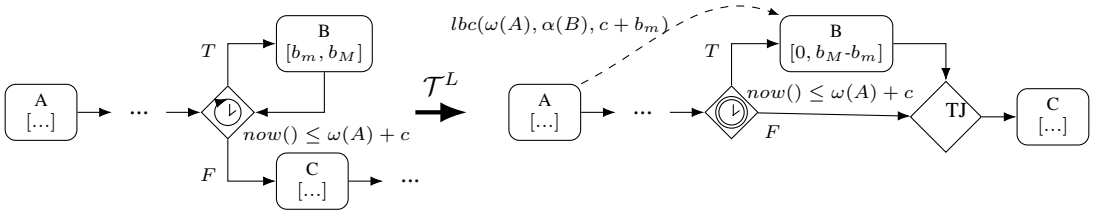


Figure 4. Transformation of a t-loop with example atomic condition $now() \leq \omega(A) + c$ into t-split.

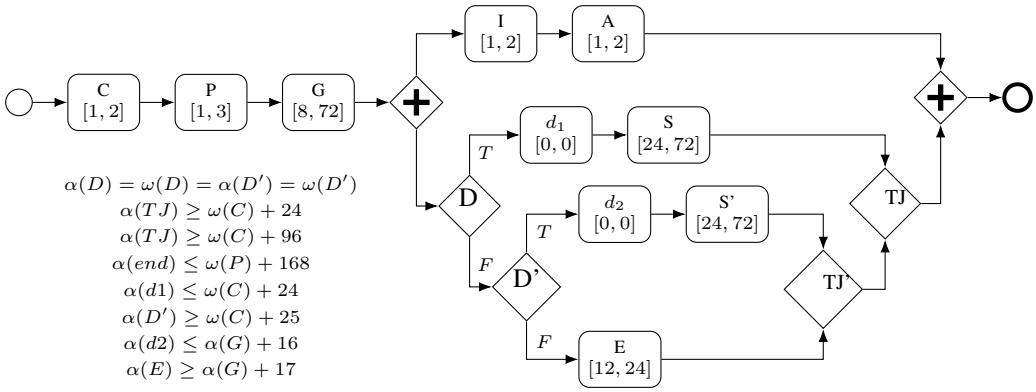


Figure 5. Transformed process model for the process in the motivating example. Gateways marked with "D", "D"' represent d-split nodes. The model can be directly translated into a CSTNUD for the DC check.

due to the advanced t-split condition, any event prior to the t-split execution may be referenced instead of only the process start, with the consequence that here we have temporal constraints potentially involving events other than process start. In Fig. 5 we show the process without t-split nodes obtained by applying the transformations described here and the transformation from t-split to d-split to the composition in the motivating example.

The processes derived by our transformations followed by the transformation in (Eder et al., 2018) are of the same type of the processes considered in (Zavatteri, 2017). Thus, we can apply the procedures in (Zavatteri, 2017) to map the transformed process models into CSTNUDs to check dynamic controllability. A limitation of our work is that we do not support all BPMN constructs, but this allows us to focus on the proposed control structures. For limitations of the checking procedure we refer to (Zavatteri et al., 2022).

5. Conclusion

The advanced temporal control structures presented here allow a highly flexible time-aware execution of business processes. They increase the controller's control power, as the time of the condition evaluation may influence its outcome and hence the continuation of the execution. The restriction of temporal constraint networks to acyclic business process models was a major impediment for the adoption of temporal control structures. Temporal loops break new ground as they allow representing repetitive elements while allowing checking dynamic controllability by unfolding them into *finite* acyclic models, as temporal cosndions are considered in relationship to other temporal properties of processes. With this work, we offer i) control structures allowing formulating alternatives and loops controlled by temporal conditions with high expressiveness, and ii) a method to check dynamic controllability of business processes featuring these structures using existing technologies.

Here, we focused on the new control structures, their definition, semantics, and dynamic controllability. Future work includes the integration with full BPMN as well as implementation and empirical evaluation of the whole transformation and checking pipeline.

References

Allweyer, T. (2016). *Bpmn 2.0: Introduction to the standard for business process modeling*. BoD–Books on Demand.

Cheikhrouhou, S., Kallel, S., Guermouche, N., & Jmaiel, M. (2013). Toward a time-centric modeling of business processes in bpmn

2.0. *Int. Conf. Information Integration and Web-based Applications & Services*, 154.

Cheikhrouhou, S., Kallel, S., Guermouche, N., & Jmaiel, M. (2015). The temporal perspective in business process modeling: A survey and research challenges. *Service Oriented Computing and Applications*, *9*(1), 75–85.

Chesani, F., et al. (2018). Compliance in business processes with incomplete information and time constraints: A general framework based on abductive reasoning. *Fundamenta Informaticae*, *161*(1-2), 75–111.

Chhetri, M. B., Vo, Q. B., & Kowalczyk, R. (2015). Supporting temporal aspects of sla establishment in auto slam framework. *Services Computing (SCC), 2015 IEEE International Conference on*, 234–241.

Combi, C., Gozzi, M., Posenato, R., & Pozzi, G. (2012). Conceptual modeling of flexible temporal workflows. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *7*(2), 1–29.

Combi, C., Oliboni, B., & Zerbato, F. (2019). A modular approach to the specification and management of time duration constraints in bpmn. *Information Systems*, *84*, 111–144.

Combi, C., & Posenato, R. (2010). Towards temporal controllabilities for workflow schemata. *2010 17th International Symposium on Temporal Representation and Reasoning*, 129–136.

Daniel, F., & Pernici, B. (2006). Insights into web service orchestration and choreography. *International Journal of E-Business Research (IJEBR)*, *2*(1), 58–77.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial intelligence*, *49*(1-3), 61–95.

Dumas, M., La Rosa, M., Mendling, J., Mäesalu, R., Reijers, H. A., & Semenenko, N. (2012). Understanding business process models: The costs and benefits of structuredness. *Advanced Information Systems Engineering*, 31–46.

Eder, J., Franceschetti, M., & Köpke, J. (2018). Controllability of orchestrations with temporal sla: Encoding temporal xor in cstnud. *Proc. 20th Int. Conf. Information Integration and Web-based Applications & Services*, 234–242.

Eder, J., Gruber, W., & Panagos, E. (2000). Temporal modeling of workflows with conditional execution paths. *Database and Expert Systems Applications*, 243–253.

Eder, J., Panagos, E., Pozewaunig, H., & Rabinovich, M. (1999). *Time management in workflow systems*. Springer.

Eder, J., & Franceschetti, M. (2020). Time and business process management: Problems, achievements, challenges (invited talk). *27th Int. Symposium on Temporal Representation and Reasoning*.

Franceschetti, M., & Eder, J. (2019). Towards checking dynamic controllability of processes with temporal loops. *ER Forum*, 1–14.

Franceschetti, M., & Eder, J. (2021). Determining temporal agreements in cross-organizational business processes. *Information and Computation*, *281*, 104792.

Hashmi, M., Governatori, G., Lam, H., & Wynn, M. T. (2018). Are we done with business process compliance: State of the art and challenges ahead. *Knowl. Inf. Syst.*, *57*(1), 79–133.

Lanz, A., Posenato, R., Combi, C., & Reichert, M. (2013). Controllability of time-aware processes at run time. *On the Move: OTM 2013 Conferences*, 39–56.

Lanz, A., Weber, B., & Reichert, M. (2014). Time patterns for process-aware information systems. *Requir. Eng.*, *19*(2), 113–141.

Pereira, J. L., & Varajao, J. (2017). The temporal dimension of business processes-dealing with time constraints. *Procedia computer science*, *121*, 1034–1038.

Pichler, H., Eder, J., & Ciglic, M. (2017). Modelling processes with time-dependent control structures. *Conceptual Modeling*, 50–58.

Posenato, R. (2022). Cstnu tool: A java library for checking temporal networks. *SoftwareX*, *17*, 100905.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences*, *280*, 218–238.

van Der Aalst, W., Ter Hofstede, A. H., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distrib. and par. databases*, *14*(1), 5–51.

Watahiki, K., Ishikawa, F., & Hiraishi, K. (2011). Formal verification of business processes with temporal and resource constraints. *Int. conf. on systems, man, and cybernetics*, 1173–1180.

Zavatteri, M. (2017). Conditional simple temporal networks with uncertainty and decisions. *LIPIcs-Leibniz International Proceedings in Informatics*, *90*.

Zavatteri, M., Rizzi, R., & Villa, T. (2022). Dynamic controllability of temporal networks with instantaneous reaction. *Information Sciences*, *613*, 932–952.