Analyzing the Impact of Process Change Operations on Time-Aware Processes

Andreas Lanz*, Manfred Reichert*

Institute of Databases and Information Systems, Ulm University, Germany

Abstract

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

Keywords: time-aware processes, dynamic process change, process flexibility, process-aware information system

1. Introduction

Time is a crucial factor regarding the support of various business processes [1]. Moreover, in many application areas (e.g., patient treatment, automotive engineering), the proper handling of *temporal constraints* is vital in order to successfully execute and complete processes [2, 3, 1]. However, contemporary process-aware information systems (PAIS) lack a more sophisticated support of such *time-aware business processes* [1]. To remedy this drawback, the proper integration of temporal constraints with both the design and run-time components of a PAIS has been identified as a key challenge [2, 3, 4].

As a prerequisite for robust process execution in PAISs, the executable *process models* must be *sound* [5]. Moreover, in the context of *time-aware process models*, i.e., process models enriched with temporal constraints, the *consistency* of the temporal constraints must be ensured [6, 3, 4]. Checking consistency of time-aware process models at design-time has been extensively studied in literature [6, 2, 7]. By contrast, only little attention has been paid to the proper run-time support of time-aware processes [4]. During run-time, the temporal consistency of process instances needs to be continuously monitored and re-checked to avoid constraint violations. Particularly, note that activity durations and deadlines may be specific to the enacted process instance and solely become known during run-time [4].

As a particular challenge, temporal constraints cannot be considered in isolation, but might interact with each other. Hence, complex algorithms are required for checking the temporal consistency of a process model [4, 8]. However, at run-time respective calculations should be reduced to a minimum to ensure

*Corresponding author

Email addresses: andreas.lanz@uni-ulm.de (Andreas Lanz), manfred.reichert@uni-ulm.de (Manfred Reichert)

scalability of the PAIS [4]. Otherwise, a run-time support of time-aware processes will not be possible at the presence of a large number of process instances.

As another challenge, time can neither be slowed down nor stopped. Accordingly, time-aware processes need to be *flexible* to cope with unforeseen events or delays during run-time [9]. For example, it is common that deadlines are re-scheduled or temporal constraints are dynamically modified in order to successfully complete a process instance being in trouble. Moreover, in certain scenarios the instances of time-aware processes must be structurally changed (e.g., by moving, deleting, or inserting activities) to be able to meet a particular deadline. In the context of such *dynamic process changes*, we must re-ensure that the resulting process instances are sound and temporally consistent. While soundness has been extensively studied in literature [10, 11, 5], this work shows how temporal consistency of a time-aware process instance can be efficiently ensured in the context of dynamic changes. Furthermore, we analyse the effects, changes have on the temporal constraints of the respective process instance. In particular, we show how the results of this analysis can be utilized to significantly reduce the complexity when applying multiple change operations. For example, the latter becomes crucial in the context of process evolution, where a possibly large set of process instances needs to be migrated on-the-fly to a changed process model [5].

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

2. Related Work

In literature, there exists considerable work on managing temporal constraints for business processes [6, 2, 7, 4, 12]. The focus of these approaches is on design-time issues like the modeling and verification of time-aware processes. By contrast, only few approaches consider execution issues of time-aware processes [3, 4]. In particular, none of the latter considers dynamic changes in this context.

Most approaches dealing with the verification of time-aware processes use a specifically tailored time model to check for the temporal consistency of process models. This becomes necessary since the interdependencies between the various temporal constraints of a process model can be quite complex and cannot be suitably captured in the respective process model. A specific conceptual model for temporal constraints is defined in [12]. In turn, [3, 7] use an extended version of the *Critical Path Method* (CPM) known from project planning. *Simple Temporal Networks* (STN) are used in [6] as basic formalism, whereas [4] suggests using *Conditional Simple Temporal Networks with Uncertainty* for checking the controllability of process models, i.e., a more restrictive form of temporal consistency. This paper relies on *Conditional Simple Temporal Networks* (CSTN), an extension of STN that allows for the proper handling of exclusive choices [8].

In [1], we presented 10 empirically evidenced time patterns (TP), that represent temporal constraints relevant in the context of time-aware processes (cf. Table 1). In particular, time patterns facilitate the comparison of existing approaches based on a universal set of notions with well-defined semantics [13]. Moreover, [13, 1] elaborated the need for a proper run-time support of the time patterns and time-aware processes.

Dynamic process changes were extensively studied in the past. Particularly, there exists considerable work on ensuring structural and behavioural soundness in the context of dynamic process instance changes [10, 11]. Further, [14] presents an overview of frequently used patterns for changing process models whose semantics is described in [11]. Finally, a comprehensive survey of approaches enabling dynamic changes is provided in [5]. To the best of our knowledge, [9] is the only work considering dynamic changes in the context of time-aware processes. As opposed to this paper, however, [9] only provides a high level discussion of the different aspects to be considered when changing time-aware process instances, temporal consistency being one of them.

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

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

Category III: Variability	
TP8	Time-dependent Variability

Category IV: Recurrent Process Elements	
TP9	Cyclic Elements
TP10	Periodicity

Table 1: Process Time Patterns TP1 – TP10 [1]

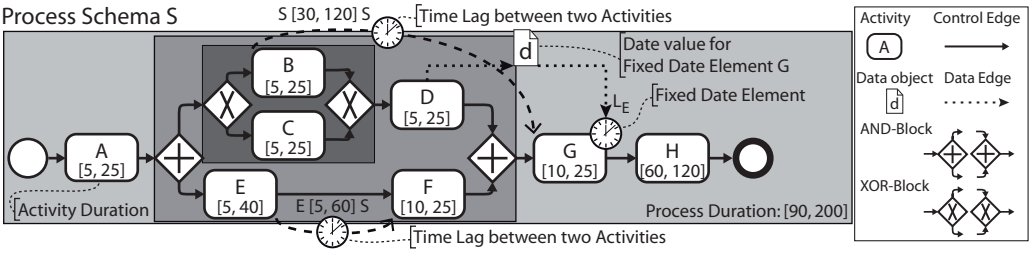


Figure 1: Core Concepts of a Time-Aware Process Model

3. Basic Notions

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

3.1. Time-aware Processes

For each business process exhibiting temporal constraints, a *time-aware process schema* needs to be defined (cf. Figure 1). In our work, a process schema corresponds to a *process model*; i.e., a directed graph, that comprises a set of *nodes*—representing *activities* and *control connectors* (e.g., Start-/End-nodes, XORsplits, or ANDjoins)—as well as a set of *control edges* linking these nodes and specifying precedence relations between them. We assume that process models are well structured [5], e.g., sequences, and branchings (i.e., parallel and exclusive choices) are specified in terms of nested blocks with unique start and end nodes of same type. These blocks—also known as SESE regions [15]—may be arbitrarily nested, but must not overlap; i.e., their nesting must be regular [16]. Figure 1 depicts an example of a well structured process model with the grey areas indicating respective blocks. Each process model contains a unique start and end node, and may be composed of control flow patterns like [17]: sequence, parallel split (ANDsplit), synchronization (ANDjoin), exclusive choice (XORsplit), and simple merge (XORjoin) (cf. Figure 1). Note that we do not consider loops in this paper. However, in the context of time-aware processes a loop may be mapped to a set of nested XOR blocks [4].

In addition, a process model contains *data objects* as well as *data edges* linking activities with data objects. More precisely, a data edge either represents a read or write access of the referenced activity to the referred data object.

At run-time, *process instances* may be created and executed according to the defined process model. We assume that a process instance is logically represented by a clone of the respective process model augmented with instance-specific information. In turn, *activity instances* represent executions of single process steps (i.e., activities) of such a process instance.

If a process model contains XOR-blocks, uncertainty is introduced since not all instances perform exactly the same set of activities. The concept of *execution path* allows us to identify which activities and control connectors are actually performed during one execution of a process instance. Particularly, given a process model, an execution path denotes a connected maximal subgraph of the process model containing its start and end node, in which all XORsplit connectors have exactly one branch. An execution path can be also briefly described by a string containing the activity identifiers of the execution path sorted with respect to

their execution order and separated by a dash if the order is sequential or by a vertical bar if it is parallel [2]. As example, considering the process model from Figure 1, the string A-((B-D)|(E-F))-G-H represents an example of an execution path, where A is followed by a parallel execution of two sequential paths (B-D) (i.e., , for the XORsplit the upper path is selected) and (E-F); then, G and H are sequentially executed. Note that the set of execution paths may have an exponential cardinality with respect to the number of XOR blocks in the process model.

We base our work on the 10 time patterns (TP) we presented in [1] (cf. Section 2). To set a focus, this work specifically considers the patterns being most relevant in practice [1]; i.e., *time lags between two activities* (TP1), *durations of activities and processes* (TP2), and *fixed date elements of activities* (TP4). In detail:

An **activity durations** (TP2) defines the minimum and maximum time span allowed for executing a particular activity (or node, in general), i.e., the time span between start and completion of the activity [1]. We assume that each activity has an assigned duration. Activity durations are described in terms of minimum and maximum values $[d_{min}, d_{max}]$ where $0 \leq d_{min} \leq d_{max}$. Since control connectors are automatically performed, one may assume that they have a fixed duration defined by the PAIS (e.g., $[0, 1]$).

Process durations (TP2) represent the time span allowed for executing an instance of the process model, i.e., the time span between start and completion of the process instance [1]. Again, a process duration is described in terms of minimum and maximum values $[d_{min}, d_{max}]$ where $0 \leq d_{min} \leq d_{max}$.

Time lags between two activities (TP1) restrict the time span allowed between the starting and/or ending instants of two arbitrary activities of a process model [1]. Such a time lag may not only be defined between directly succeeding activities, but between any two activities that may be conjointly executed in the context of a particular process instance, i.e., the activities must not belong to exclusive branches. In Figure 1, a time lag is visualized through a dashed edge with a clock symbol between the source and target activity. The label of the edge specifies the constraint according to the following template: $\langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle$; $\langle I_S \rangle \in \{S, E\}$ and $\langle I_T \rangle \in \{S, E\}$ mark the instant (i.e., starting or ending) of the source and target activity the time lag applies to; e.g., $\langle I_S \rangle = S$ marks the starting instant of the source activity and $\langle I_T \rangle = E$ the ending instant of the target activity. In turn, $[t_{min}, t_{max}]$ $(-\infty \leq t_{min} \leq t_{max} \leq \infty)$ represents the range allowed for the time span between instants $\langle I_S \rangle$ and $\langle I_T \rangle$. In particular, time lags may be used to specify minimum delays and maximum waiting times between succeeding activities. Finally, note that a control edge implicitly represents an $E[0, \infty]S$ time lag between its source and target activity, i.e., the target activity may only be started after completing the source activity.

Fixed date elements (TP4) refer to activities and allow restricting their execution in relation to a specific date [1], e.g., a fixed date element may define that the activity must not be started before or must be completed by a particular date.[1] Generally, the value of a fixed date element is specific to a process instance, i.e., it is not known before creating the process instance or even becomes known only during run time. Therefore, the respective date of a fixed date element is stored in a data object. When evaluating the fixed date element during run-time, the current value of the respective data object is retrieved [13]. Figure 1 visualizes a fixed date element through a clock symbol attached to the activity. In this context, label $\langle D \rangle \in \{E_S, L_S, E_E, L_E\}$ represents the activity's earliest start date (E_S), latest start date (L_S), earliest completion date (E_E), or latest completion date (L_E), respectively.

Figure 1 shows an example of a process model exhibiting temporal constraints. Even though we use the notation defined by BPMN for illustration purpose, the approach described in the following is not BPMN-specific. Further note that, although some of the symbols used for visualizing the temporal constraints resemble BPMN timer events, their semantics is quite different and should not be mixed up. As can be easily verified, in Figure 1 all activities have a corresponding activity duration. The duration of activity A, for example, expresses that A has a minimum duration of 5 and a maximum duration of 25. Between B and G there is a time lag described by $S[30, 120]S$, i.e., between the start of B and the start of G there must be a minimum delay of 30 time units and a maximum waiting time of 120 time units. Additionally, there is a time lag between E and F. Next, G has a fixed date element attached to it, whereby label L_E indicates that

[1] Fixed date elements are often referred to as "deadlines". However, this does not completely meet the intended semantics.

the latest end date of the activity is restricted by the temporal constraint. In turn, the date of the fixed date element is provided by activity D through data object d.

3.2. Temporal Consistency of Time-Aware Processes

A time-aware process model is executed by performing its activities and control connectors, thereby obeying any structural or temporal constraints of the process model. We denote a process model as *temporally consistent* if it is possible to perform all *execution paths* without violating the temporal constraints involved. Temporal consistency of a time-aware process model (as well as respective process instances) constitutes a fundamental prerequisite for its robust and error-free execution. In particular, executing a process instance whose model is not temporally consistent leads to a waste of resources [6, 3]. Therefore, for any PAIS supporting time-aware processes, a crucial task is to check temporal consistency of the process model at design-time as well as to monitor and re-check corresponding instance during run-time. This is particularly challenging since different temporal constraints might interact with each other resulting in complex interdependcies (e.g., a future deadline might restrict the duration of some or all preceding activities).

Whether or not a time-aware process model is temporally consistent can be checked by mapping it to a *conditional simple temporal network* (CSTN)—a problem known from artificial intelligence [8, 18]. In our work, we use CSTN since it allows us to exploit and reuse *checking algorithms* for a well founded model for representing temporal constraints. Finally, CSTN allows capturing the complex interdependencies between constraints, which cannot be suitably captured in time-aware process models.

Definition 1 (Conditional Simple Temporal Network)**.** *A* Conditional Simple Temporal Network *(CSTN) is a 6-tuple* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$*, where:*[2]

- $\mathcal{T}$ *is a set of real-valued variables, called* time-points*;*
- P *is a finite set of* propositional letters *(or propositions);*
- $L : \mathcal{T} \rightarrow P^*$ *is a function assigning a* label *to each time-point in* $\mathcal{T}$*; a* label *is any (possibly empty) conjunction of (positive or negative) letters from* P*.*[3]
- $\mathcal{C}$ *is a set of* labeled simple temporal constraints *(*constraint *in the following); each constraint* $c_{XY} \in \mathcal{C}$ *has the form* $c_{XY} = \langle [x, y]_{XY}, \beta \rangle$*, where* $X, Y \in \mathcal{T}$ *are any time-points,* $-\infty \leq x \leq y \leq \infty$ *are any real numbers, and* $\beta \in P^*$ *is a label.*
- $\mathcal{OT} \subseteq \mathcal{T}$ *is a set of* observation time-points*;*
- $O : P \rightarrow \mathcal{OT}$ *is a bijection that associates a unique observation time-point to each propositional letter from* P*.*

Time-points represent instantaneous events that may be, for example, associated with the start or end of activities. In turn, *observation time-points* represent the time point at which relevant information for the execution of the CSTN is acquired, i.e., it represents the time-point a decision regarding possible execution paths is made. More formally, when executing observation time-point P, the truth-value of the associated proposition (i.e., $O^{-1}(P)$) is determined (observed in CSTN jargon). A *constraint* $c_{XY} = \langle [x, y]_{XY}, \beta \rangle$ expresses that the time span between time-points X and Y must be at least x and at most y, i.e., $Y - X \in [x, y]$. The *label* attached to each time-point and constraint, respectively, indicates the different possible executions of the CSTN, i.e., a particular time-point or constraint will be only considered if the corresponding label is satisfiable in the respective instance. Figure 2 depicts the CSTN corresponding to the process model from Figure 1.

The solution to a CSTN can be defined as follows [18]:

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

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

[2] For a more complete definition and a characterization see [18].

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

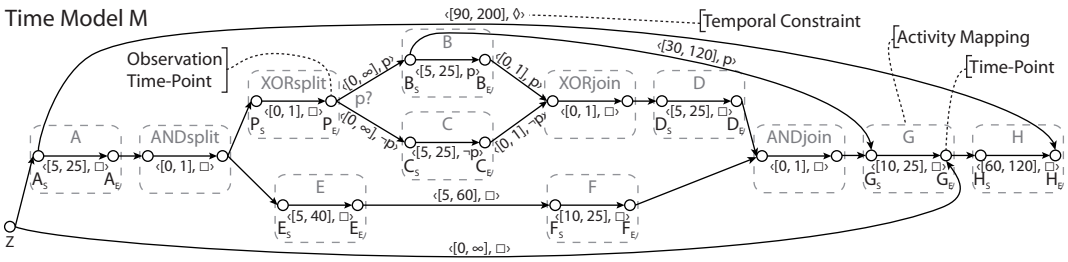


Figure 2: CSTN Representation of the Process Model from Figure 1

We denote the CSTN corresponding to a time-aware process model as its *time model*. The required mapping can roughly be described as follows:[4] First, the control flow of the process model is mapped to a CSTN as illustrated in the left part of Figure 3. Particularly, each control flow element implicitly represents a temporal constraint, e.g., a control edge is equivalent to an $E[0, \infty]S$ time lag (cf. Figure 3). Each activity, ANDsplit, ANDjoin, and XORjoin n_i is represented as a pair of time-points N_{iS} and N_{iE}, corresponding to the starting and ending instant of the respective node (cf. Figure 3). In turn, for an XORsplit, the ending instant (i.e., N_{iE}) is represented by an observation time-point (cf. Figure 3). Next, a constraint $\langle [d_{min}, d_{max}]_{N_{iS}N_{iE}}, \boxdot \rangle$ is added between N_{iS} and N_{iE} representing the duration $[d_{min}, d_{max}]$ of the respective node. Furthermore, for any control edge between nodes n_i and n_j, a constraint $\langle [0, \infty]_{N_{iE}N_{jS}}, \boxdot \rangle$ is added between the time-points representing the ending instant of n_i and starting instant of n_j (cf. Figure 3). If the source of the control edge corresponds to an XORsplit, in addition, the label of the constraint is augmented by proposition $p = O^{-1}(P)$. The latter represents the decision made at the corresponding observation time-point P, i.e., the label of the constraint $\langle [0, \infty]_{N_{iE}N_{jS}}, \beta \rangle$ belonging to the "`true`"-branch is set to βp and the label of the "`false`"-branch to $\beta \neg p$ (cf. Figure 3).[5] Further, the labels of all constraints and time-points corresponding to activities, connectors and control edges in the XOR-block are augmented by either p or $\neg p$ depending on the branch they belong to.

Next, temporal constraints are mapped to the CSTN as depicted in the right part of Figure 3. A time lag $\langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle$ corresponds to a constraint $\langle [t_{min}, t_{max}]_{N_{i\langle I_S \rangle} N_{j\langle I_T \rangle}}, L(N_{i\langle I_S \rangle}) \wedge L(N_{j\langle I_T \rangle}) \rangle$ between the two time-points representing the respective instants of nodes n_i and n_j (cf. Figure 3). In turn, a fixed date element is initially represented as a constraint $\langle [0, \infty]_{Z N_{\langle D \rangle}}, L(N_{\langle D \rangle}) \rangle$ with Z being a special time-point representing time "0" (cf. Figure 3). During run-time, value $[0, \infty]$ of the constraint will be updated according to the actual fixed date chosen. Finally, process duration $[d_{min}, d_{max}]$ is represented as constraint $\langle [d_{min}, d_{max}]_{N_{0S}N_{kE}}, \boxdot \rangle$ between the time-points representing the starting instant N_{0S} of the first and the one representing the ending instant N_{kE} of the last node of the process.

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

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

Definition 3 (Temporal Consistency)**.** *A CSTN $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ is called* weakly consistent *iff for each scenario s_P at least one viable solution exists [8].*

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

When executing a time-aware process model, temporal consistency of the respective instances needs to be continuously monitored and re-checked. For this purpose, the *minimal network* of a CSTN[6] must be determined.

[4]For further details we refer to [4], where we show how time-aware processes can be transformed to CSTNU—a special kind of CSTN.

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

[6]The minimal network of a CSTN is also called its *dispatchable form*.

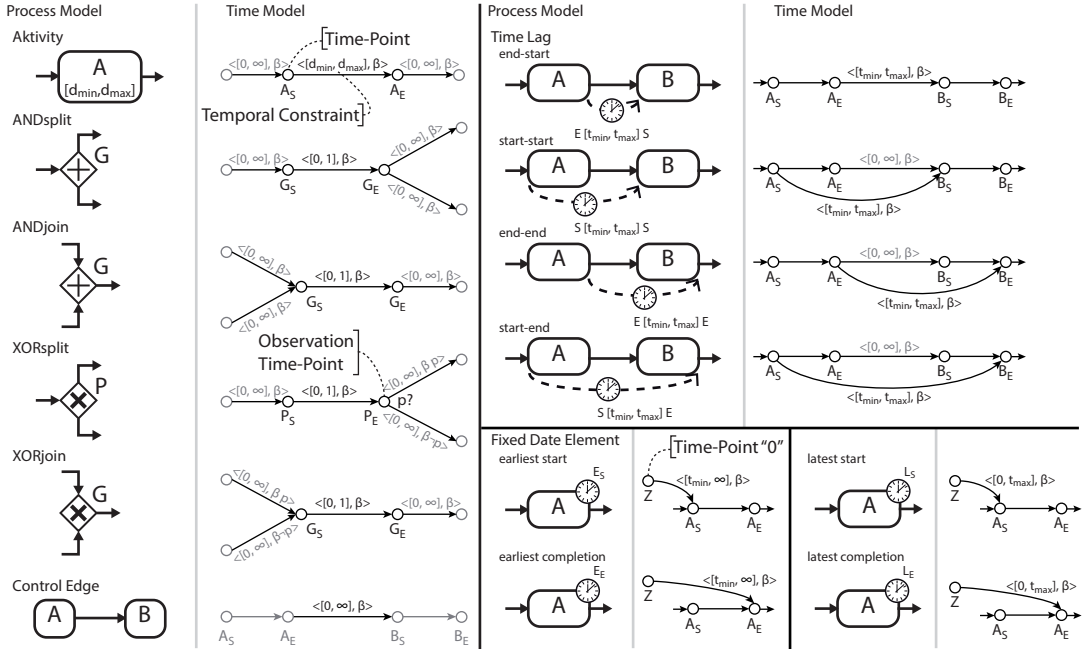


Figure 3: Process Modeling Elements and their Mapping to CSTN

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

For any CSTN S a minimal network exists iff S is *weakly consistent*. In particular, such a minimal network provides a restricted set of constraints: As long as the value of each time-point is consistent with all constraints referring to it whose labels are still satisfiable, we can guarantee that the entire CSTN is weakly consistent. Besides *explicit constraints* $c \in \mathcal{C}$ we obtain when mapping the process model to the CSTN, the minimal network contains *implicit constraints* between any pair of time-points that may occur in the same execution path. Note that these implicit constraints represent the effects the explicit constraints have on the overall CSTN (i.e., they represent interdependencies between explicit constraints). The implicit constraints are derived from the explicit ones when determining the minimal network. How to determine the latter is described in [8]. In the following, we denote the time model resulting from the mapping of the process model to a CSTN as *base time model* and its minimal network as *minimal time model* of the process model.

As example consider Figure 4 which depicts a process model, its base time model and the corresponding minimal time model. Note that for the sake of readability most implicit constraints of the minimal time model are only indicated through light grey arrows. Further note, that explicit constraints which are restricted when minimizing the time model are highlighted in green. The example also illustrates some of the possible interdependencies between the various temporal constraints of a process model. In particular, the time lag between the start of activity `B` and start of `C` restricts the maximum duration of `B` to 20; note the constraint between B_S and B_E in the minimal time model. Moreover, the time lags between `A` and `C` and `B` and `C` introduces an additional interdependency between activities `A` and `B` (the respective implicit temporal constraint is highlighted in red in Figure 4). In particular, although activities `A` and `B` are seemingly temporally unrelated, `B` may be started the earliest 5 time units after the start of `A`. Otherwise, some temporal constraints of the process model cannot be satisfied. As a consequence, `B` may be started the earliest 5 time units after completion of the ANDsplit node. Furthermore, `B` must be started the latest 30 time units after the start of `A`. Note that representing all these interdependencies of different temporal constraints as part of the process model is not feasible as they can be quite numerous. Particularly, this would render the process model unreadable.

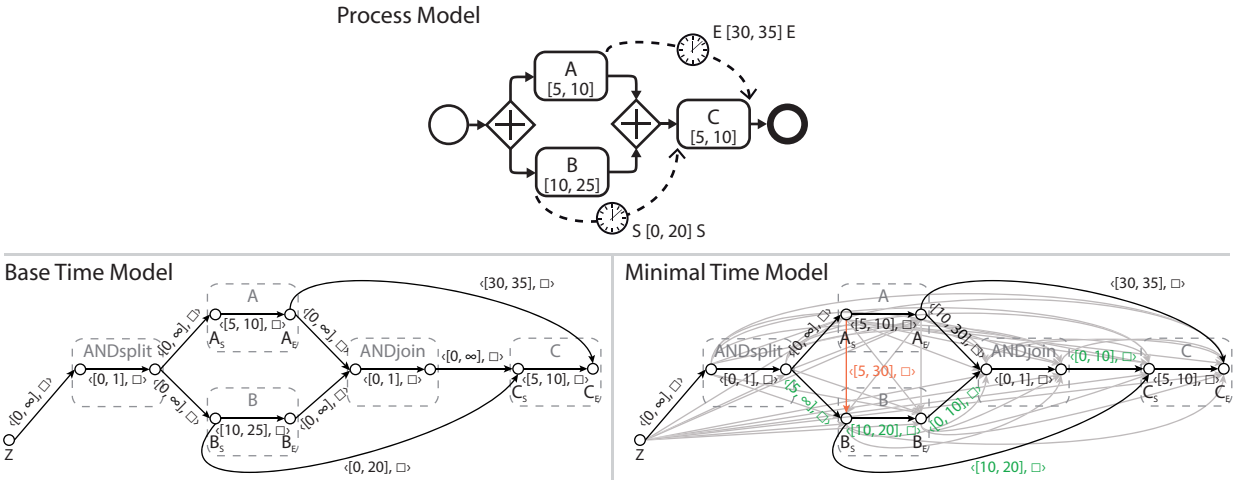


Figure 4: Basic and Minimal Time Model

When executing a process instance, the two time models created at design-time are cloned. These clones are then kept up-to-date with the actual temporal state of the process instance (e.g., activity start and completion times) and used to monitor and re-check temporal consistency of the process instance [4].

4. Change Operations for Time-aware Processes

Standard change patterns adapting process models without temporal constraints have been extensively studied in literature [5]. This section discusses how respective change operations may be transferred to time-aware processes. Section 4.1 presents the change operations applicable to time-aware processes. Section 4.2 then provides an in-depth discussion of these change operations and shows how they can be extended to ensure temporal consistency of a changed process model.

4.1. Basic Change Operations

When changing a process schema or process instance, respectively, or—more generally—when changing a process model, the soundness of the latter must be ensured. To achieve this, our framework abstracts from low-level *change primitives* (e.g., adding an edge or node) to higher-level *change operations* with well-defined pre-/post-conditions [5]. When being applied to a sound process model, such a high-level change operation (e.g., inserting a node in serial between two succeeding nodes) guarantees that the resulting process model will be structurally and behaviourally sound as well [5]. The upper part of Table 2 gives an overview of the most important change operations required for structurally modifying a process model. Note that these change operations may be combined to realize more complex change patterns [14] (e.g., move activity). We extend the set of structural change operations by a new set of change operations enabling us to modify the temporal constraints of a process model as well, e.g., inserting a time lag (see the bottom of Table 2). Altogether, these change operations allow changing a time-aware process model, while guaranteeing structural and behavioural soundness of the resulting process model.

4.2. Applying Change Operations to Time-aware Processes

When modifying the model of a time-aware process, it must be ensured that the resulting process model is temporally consistent. This section defines basic criteria ensuring that the application of a change operation does not result in a temporally inconsistent process model. We further analyze the local impact a particular change operation has on the temporal properties of the respective process model, i.e., its temporal constraints.

If a change operation is applied to a process instance, additional state-specific pre- and post-conditions need to be met [5]. These are not considered in the following since they apply to time-aware processes as well. Further, note that any time-related instance-specific information is captured in the corresponding time

Operation	Description
Control Flow	
$InsertSerial(n_1, n_2, n_{new}, [d_{min}, d_{max}])$	Inserts node n_{new} with duration $[d_{min}, d_{max}]$ between directly succeeding nodes n_1 and n_2.
$InsertPar(n_1, n_2, n_{new}, [d_{min}, d_{max}])$	Inserts node n_{new} with duration $[d_{min}, d_{max}]$ as well as an AND block surrounding the SESE block defined by n_1 and n_2.
$InsertCond(n_1, n_2, n_{new}, [d_{min}, d_{max}], c)$	Inserts node n_{new} with duration $[d_{min}, d_{max}]$ and condition c as well as an XOR block between succeeding nodes n_1 and n_2.
$InsertBranch(g_1, g_2, c)$	Inserts an empty branch with condition c between XORsplit g_1 and XORjoin g_2.
$DeleteActivity(n)$	Deletes activity n.
Temporal Constraints	
$InsertTimeLag(n_1, n_2, type_{tl}, [t_{min}, t_{max}])$	Inserts a time lag $[t_{min}, t_{max}]$ between nodes n_1 and n_2. Thereby, $type_{tl} \in$ {start-start, start-end, end-start, end-end} describes whether the time lag is inserted between the start of the two activities, the start of n_1 and the end of n_2, the end of n_1 and the start of n_2, or the end of the two activities.
$InsertFDE(n, type_{fde})$	Adds a fixed date element of type $type_{fde} \in \{E_S, L_S, E_E, L_E\}$ to node n.
$DeleteTimeLag(n_1, n_2, type_{tl})$	Deletes the time lag of type $type_{tl}$ between nodes n_1 and n_2.
$DeleteFDE(n, type_{fde})$	Deletes any fixed date element of type $type_{fde}$ from node n.

Table 2: Basic Change Operations

model (cf. Section 3.2). In particular, we will show that it is sufficient to only consider the current minimal time model of the process instance.

4.2.1. Serial Activity Insertion.

As first change operation we consider $InsertSerial(n_1, n_2, n_{new}, [d_{min}, d_{max}])$. It allows inserting a node n_{new} with duration $[d_{min}, d_{max}]$ between two directly succeeding nodes n_1 and n_2 (cf. Figure 5). In terms of change primitives, this can be realized by deleting the control edge between nodes n_1 and n_2, followed by adding node n_{new} with duration $[d_{min}, d_{max}]$ to the process model and properly connecting it to n_1 and n_2 by adding two new control edges (cf. Figure 5 and Table 3). Regarding the temporal properties of the resulting process model, one can observe that the insertion of n_{new} will first and foremost increase the minimum time distance between n_1 and n_2 to d_{min}. By contrast, the maximum distance between the two nodes is not affected by the change as the newly added control connectors do not constrain it. Accordingly, if for the minimal time model the minimum duration d_{min} is compliant with any implicit or explicit constraint $\langle [c_{min}, c_{max}]_{N_1 E N_2 S}, \beta \rangle$ between the ending instant of n_1 and the starting instant of n_2 (i.e., $d_{min} \leq c_{max}$), the node insertion will not affect temporal consistency of the process model.[7] Remember that each value of each constraint in the minimal time model is part of at least one solution (cf. Definition 4), i.e., one viable execution of the process model. Consequently, the time-aware process model is still temporally consistent.

After inserting the node into the process model, the mapping of this node and the control edges must be added to the time models as well (i.e., the base time model and minimal time model). Furthermore, the minimal time model must be locally adapted in order to properly cover the changes. In particular, the constraint between the ending instant of n_1 and the starting instant of n_2 must be updated to $[\max\{c_{min}, d_{min}\}, c_{max}]$ to consider the new minimum distance between the two nodes (cf. Figure 5), i.e., certain values permitted by the old constraint might no longer be part of any viable solution. It further becomes evident that the constraints corresponding to the two control edges must be initialized to $[0, c_{max} - d_{min}]$ (cf. Figure 5). Algorithm 1 defines the pre- and post-conditions for applying change operation *InsertSerial* to a process model.

[7] Note that any implicit constraint $\langle [c_{min}, c_{max}]_{N_1 E N_2 S}, \beta \rangle$ is always at least as restrictive as any explicit time lag $E[t_{min}, t_{max}]S$ between n_1 and n_2.

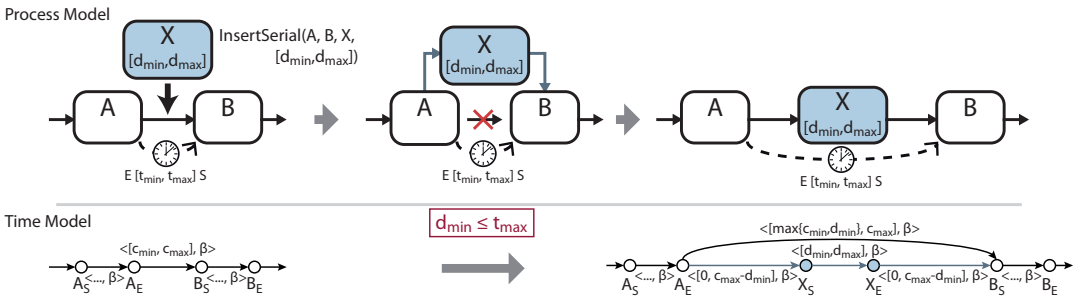


Figure 5: Change Operation: Insert Serial

Algorithm 1: InsertSerial$(\mathbf{n_1}, \mathbf{n_2}, \mathbf{n_{new}}, [\mathbf{d_{min}}, \mathbf{d_{max}}])$	
Pre	$succ(n_1) = n_2$, $\forall \langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle \in \mathcal{C} : c_{max} \geq d_{min}$
Init	$\gamma = L(N_{1E}) \wedge L(N_{2S})$
Post	// Update process model: $RemoveEdge(n_1, n_2)$, $AddNode(n_{new}, [d_{min}, d_{max}], Activity)$, $AddEdge(n_1, n_{new})$, $AddEdge(n_{new}, n_2)$
	// Update basic and minimal time model: $AddTimePoint(N_{newS}, \gamma)$, $AddTimePoint(N_{newE}, \gamma)$, $AddConstraint(N_{newS}, N_{newE}, [d_{min}, d_{max}], \gamma)$, $AddConstraint(N_{1E}, N_{newS}, [0, \infty], \gamma)$, $AddConstraint(N_{newE}, N_{2S}, [0, \infty], \gamma)$,
	// Adapt minimal time model: $\forall \langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle \in \mathcal{C}$: $AddConstraint(N_{1E}, N_{newS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(N_{newE}, N_{2S}, [0, c_{max} - d_{min}], \beta)$, $UpdateConstraint(N_{1E}, N_{2S}, [\max\{c_{min}, d_{min}\}, c_{max})], \beta)$

Table 3: Algorithm 1: InsertSerial

After adding the node to the process model, the mapping of this node and the control edges must be added to the time models as well (i.e., the base time model and minimal time model). Further, the minimal time model must be locally adapted to properly cover the changes. In particular, the constraint between the ending instant of n_1 and the starting instant of n_2 must be updated to $[\max\{c_{min}, d_{min}\}, c_{max}]$ in order to consider the new minimum distance between the two nodes (cf. Figure 5), i.e., certain values permitted by the old constraint might no longer be part of any viable solution. It further becomes evident that the constraints corresponding to the two control edges must be initialized to $[0, c_{max} - d_{min}]$ (cf. Figure 5). Algorithm 1 defines the pre- and post-conditions for applying change operation *InsertSerial* to a process model.

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

4.2.2. Parallel Activity Insertion.

The next change operation considered by us is operation $\boldsymbol{InsertPar}(n_1, n_2, n_{new}, [d_{min}, d_{max}])$. It inserts node n_{new} together with an ANDsplit- and ANDjoin-gateway surrounding the SESE-region defined by n_1 and n_2 (cf. Figure 6 and Table 4). This is achieved by serially inserting ANDsplit g_s between n_1 and its predecessor and ANDjoin g_j between n_2 and its successor (cf. *InsertSerial*-operation). Next, node n_{new} with duration $[d_{min}, d_{max}]$ is added to the process model and properly connected to g_s and g_j by adding two new control edges (cf. Figure 6). Concerning temporal constraints, we again observe that inserting the node solely increases the minimum time distance between the predecessor n_p of n_1 and the successor n_s of n_2. Consequently, if the minimum duration d_{min} is compliant with any implicit (or explicit) constraint $\langle [c_{min},$

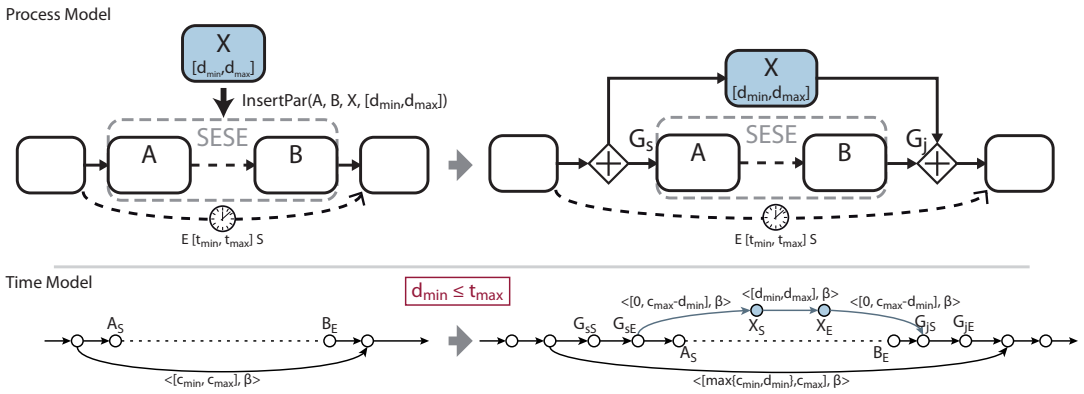


Figure 6: Update Operation: Insert Parallel

Algorithm 2: InsertPar$(\mathbf{n_1}, \mathbf{n_2}, \mathbf{n_{new}}, [\mathbf{d_{min}}, \mathbf{d_{max}}])$	
Pre	(n_1, n_2) is SESE-region, $\forall \langle [c_{min}, c_{max}]_{N_{pE}N_{sS}}, \beta \rangle \in \mathcal{C} : c_{max} \geq d_{min}$,
Init	$n_p = pred(n_1)$, $n_s = succ(n_2)$, $\gamma = L(N_{1E}) \wedge L(N_{2S})$
Post	// Update process model: $AddNode(g_s, [0,1], AND)$, $AddNode(g_j, [0,1], AND)$, $RemoveEdge(n_p, n_1)$, $AddEdge(n_p, g_s)$, $AddEdge(g_s, n_1)$, $RemoveEdge(n_2, n_s)$, $AddEdge(n_2, g_j)$, $AddEdge(g_j, n_s)$, $AddNode(n_{new}, [d_{min}, d_{max}], Activity)$, $AddEdge(g_s, n_{new})$, $AddEdge(n_{new}, g_j)$,
	// Update basic and minimal time model: $AddTimePoint(G_{sS}, \gamma)$, $AddTimePoint(G_{sE}, \gamma)$, $AddConstraint(G_{sS}, G_{sE}, [0,1], \gamma)$, $AddTimePoint(G_{jS}, \gamma)$, $AddTimePoint(G_{jE}, \gamma)$, $AddConstraint(G_{jS}, G_{jE}, [0,1], \gamma)$, $AddConstraint(N_{pE}, G_{sS}, [0, \infty], \gamma)$, $AddConstraint(G_{sE}, N_{1E}, [0, \infty], \gamma)$, $AddConstraint(N_{2E}, G_{jS}, [0, \infty], \gamma)$, $AddConstraint(G_{jE}, N_{sS}, [0, \infty], \gamma)$, $AddTimePoint(N_{newS}, \gamma)$, $AddTimePoint(N_{newE}, \gamma)$, $AddConstraint(N_{newS}, N_{newE}, [d_{min}, d_{max}], \gamma)$, $AddConstraint(N_{pE}, N_{newS}, [0, \infty], \gamma)$, $AddConstraint(N_{newE}, N_{sS}, [0, \infty], \gamma)$, $AddConstraint(G_{sE}, N_{newS}, [0, \infty], \gamma)$, $AddConstraint(N_{newE}, G_{jS}, [0, \infty], \gamma)$
	// Adapt minimal time model: $\forall \langle [c_{min}, c_{max}]_{N_{pE}N_{sS}}, \beta \rangle \in \mathcal{C}$: $AddConstraint(N_{pE}, G_{sS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(G_{sE}, N_{newS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(N_{newE}, G_{jS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(G_{jE}, N_{sS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(N_{pE}, N_{newS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(N_{newE}, N_{sS}, [0, c_{max} - d_{min}], \beta)$, $UpdateConstraint(N_{pE}, N_{sS}, [\max\{c_{min}, d_{min}\}, c_{max}], \beta)$

Table 4: Algorithm 2: InsertPar

$c_{max}]_{N_{pE}N_{sS}}, \beta \rangle$ between the respective instants of n_p and n_s, performing the change operation does not impact consistency of the process model. Note that the added ANDsplit and ANDjoin gateways constitute silent nodes.

Again, after structurally modifying the process model, the time models needs to be adapted to reflect the change. Particularly, in the minimal time model the temporal constraint between the ending instant of n_p and the starting instant of n_s must be updated to $[\max\{c_{min}, d_{min}\}, c_{max}]$ (cf. Figure 6). Moreover, the constraints representing the newly added control edges must be initialized to $[0, c_{max} - d_{min}]$ to reflect the impact the other constraint have on the time to executed the new activity (cf. Figure 6). Algorithm 2 (cf. Table 4) formally defines these pre- and post-conditions for performing operation *InsertPar*.

At last, the changes made to the minimal time model need to be propagated to the other constraints

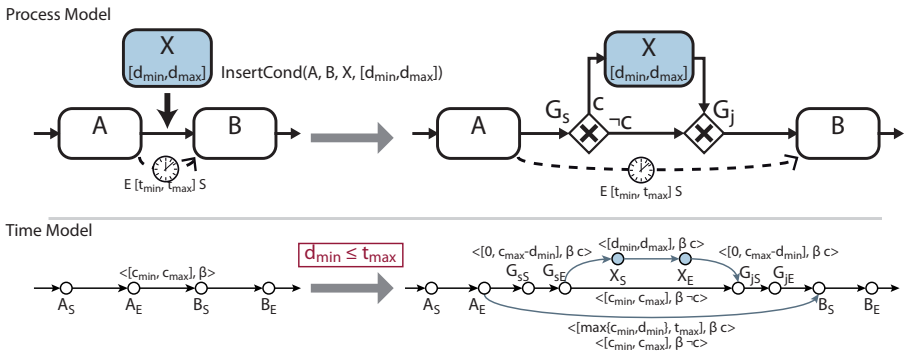


Figure 7: Change Operation: Insert Conditional

Algorithm 3: InsertCond$(\mathbf{n_1}, \mathbf{n_2}, \mathbf{n_{new}}, [\mathbf{d_{min}}, \mathbf{d_{max}}], \mathbf{c})$	
Pre	$succ(n_1) = n_2$, $\forall\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta\rangle \in \mathcal{C} : c_{max} \geq d_{min}$
Init	$\gamma = L(N_{1E}) \wedge L(N_{2S})$
Post	// Update process model: $RemoveEdge(n_1, n_2)$, $AddNode(g_s, [0,1], XOR)$, $AddNode(g_j, [0,1], XOR)$, $AddEdge(n_1, g_s)$, $AddEdge(g_s, g_j)$, $AddEdge(g_j, n_2)$, $AddNode(n_{new}, [d_{min}, d_{max}], Activity)$, $AddEdge(g_s, n_{new})$, $AddEdge(n_{new}, g_j)$, $UpdateCondition(g_s, n_{new}, c)$, $UpdateCondition(g_s, g_j, \neg c)$,
	// Update basic and minimal time model: $AddTimePoint(G_{sS}, \gamma)$, $AddObservationTimePoint(G_{sE}, c, \gamma)$, $AddConstraint(G_{sS}, G_{sE}, [0,1], \gamma)$, $AddTimePoint(N_{newS}, \gamma)$, $AddTimePoint(N_{newE}, \gamma c)$, $AddConstraint(N_{newS}, N_{newE}, [d_{min}, d_{max}], \gamma c)$, $AddTimePoint(G_{jS}, \gamma)$, $AddTimePoint(G_{jE}, \gamma)$, $AddConstraint(G_{jS}, G_{jE}, [0,1], \gamma)$, $AddConstraint(N_{1E}, G_{sS}, [0, \infty], \gamma)$, $AddConstraint(G_{jE}, N_{2S}, [0, \infty], \gamma)$, $AddConstraint(N_{newE}, G_{jS}, [0, \infty], \gamma c)$, $AddConstraint(G_{sE}, N_{newS}, [0, \infty], \gamma c)$, $AddConstraint(N_{1E}, N_{newS}, [0, \infty], \gamma c)$, $AddConstraint(N_{newE}, N_{2S}, [0, \infty], \gamma c)$, $AddConstraint(G_{sE}, G_{jS}, [0, \infty], \gamma \neg c)$,
	// Update minimal time model: $\forall\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta\rangle \in \mathcal{C}$: $AddConstraint(N_{1E}, G_{sS}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(G_{sE}, N_{newS}, [0, c_{max} - d_{min}], \beta c)$, $AddConstraint(N_{newE}, G_{jS}, [0, c_{max} - d_{min}], \beta c)$, $AddConstraint(G_{jE}, N_{2S}, [0, c_{max} - d_{min}], \beta)$, $AddConstraint(N_{1E}, N_{newS}, [0, c_{max} - d_{min}], \beta c)$, $AddConstraint(N_{newE}, N_{2S}, [0, c_{max} - d_{min}], \beta c)$, $AddConstraint(G_{sE}, G_{jS}, [c_{min}, c_{max}], \beta \neg c)$, $UpdateConstraint(N_{1E}, N_{2S}, [c_{min}, c_{max}], \beta \neg c)$, $AddConstraint(N_{1E}, N_{2S}, [\max\{c_{min}, d_{min}\}, c_{max}], \beta c)$

Table 5: Algorithm 3: InsertCond

before performing any other change operation, i.e., the minimality of the modified time model needs to be restored.

4.2.3. Conditional Activity Insertion.

Change operation $InsertCond(n_1, n_2, n_{new}, [d_{min}, d_{max}], c)$ inserts node n_{new} conditionally between succeeding nodes n_1 and n_2. This change is accomplished by first inserting XORsplit g_s and XORjoin g_j sequentially between n_1 and n_2 and then inserting n_{new} conditionally between g_s and g_j (cf. Figure 7). The transition condition of the control edge linking g_s and n_{new} is set to c and the one of the control edge linking

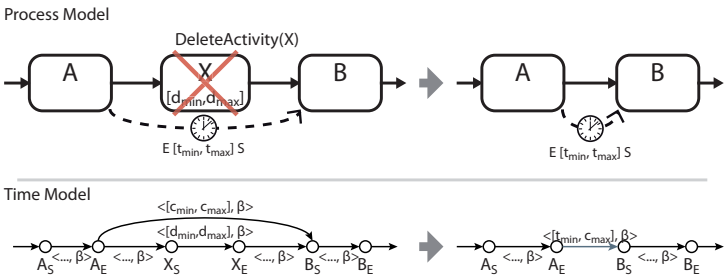


Figure 8: Update Operation: Delete Activity

g_s and g_j to $\neg c$. Note that when adding XORsplit g_s and condition c / $\neg c$ to the process model, this results in a set of additional execution paths; i.e., each execution path of the old process model, which contains n_1 and n_2, can now be mapped to two execution paths: one path with $c = false$ (i.e., $\neg c$) representing the previous execution path and one with $c = true$ representing the new one containing n_{new} between n_1 and n_2. Hence, for any execution path containing n_{new}, *InsertCond* has similar effects as *InsertSerial*. In turn, any execution path not containing n_{new} remains unchanged (except for the added XORsplit and XORjoin, that constitute silent nodes). Altogether, for operation *InsertCond* similar pre-conditions hold as for *InsertSerial* (cf. Table 3).

After changing a process model, the corresponding time model needs to be adapted by adding the mappings of the inserted elements as shown in Figure 7. In particular, note that this adds a new observation time-point G_{sE} and proposition c to the time model (cf. Section 3.2). Accordingly, the labels of the temporal constraints representing n_{new} and the two control edges connecting it with g_s and g_j, respectively, must be set to βc with β being the label of the original constraint between N_{1E} and N_{2S}. In turn, the label of the constraint corresponding to the control edge between g_s and g_j must be set to $\beta\neg c$. Finally, the constraint between the ending instant of n_1 and the starting instant of n_2 needs to be updated as follows: The label of the original constraint must be augmented by proposition $\neg c$ resulting in constraint $\langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta\neg c\rangle$. Further, another constraint $\langle [\max\{c_{min}, d_{min}\}, c_{max}]_{N_{1E}N_{2S}}, \beta c\rangle$ containing proposition c must be added between the two time-points. The latter corresponds to the case where n_{new} is executed between the two nodes. Algorithm 3 defines the pre- and post-conditions of operation *InsertCond*. After applying this operation, again the minimality of the adapted minimal time model has to be restored. This must be accomplished before performing any other change or resuming the execution of the process instance.

4.2.4. Branch Insertion.

Operation *InsertBranch* is similar to *InsertCond*, except that no node is added. Since only a control edge is added, no specific time-related pre-conditions must be satisfied.

4.2.5. Activity Deletion.

Change operation *DeleteActivity* allows deleting activities from a process model as depicted in Figure 8. Particularly, it removes activity n and replaces it by a new control edge between its predecessor and successor. From a temporal point of view, deleting an activity is always possible as temporal constraints are only removed from the time model, but no new ones are added. As only pre-condition, we require that the activity to be removed has no remaining incoming or outgoing explicit temporal constraint (i.e., time lag or fixed date element). If necessary, these have to be removed in advance using respective change operations (cf. Table 2).

Algorithm 4 (cf. Table 6) defines the respective pre- and post conditions of operation *DeleteActivity*. Unfortunately, when deleting an activity from the process schema it is not possible to restore minimality of the modified minimal time model. In particular, it is not possible to determine which of the values previously removed from the constraints when establishing minimality of the time model may now be added again. Instead it is necessary to recalculate the minimal time model from the base time model.

Algorithm 4: DeleteActivity(n)	
Pre	n has no incoming or outgoing explicit temporal constraint,
Init	$n_p = pred(n)$, $n_s = succ(n)$, t_{min} is the minimum distance of the explicit constraint between n_p and n_s if any, or $t_{min} = 0$ if there is no explicit constraint
Post	// Update process model: $RemoveEdge(n_p, n)$, $RemoveEdge(n, n_s)$, $RemoveNode(n)$, $AddEdge(n_p, n_s)$
	// Update basic time model: $RemoveConstraint(N_S, N_E)$ $\forall t \in \mathcal{T} \setminus \{N_S, N_E\} : RemoveConstraint(t, N_S), RemoveConstraint(t, N_E),$ $RemoveConstraint(N_S, t), RemoveConstraint(N_E, t)$ $RemoveTimePoint(N_S)$, $RemoveTimePoint(N_E)$, $AddConstraint(n_p, n_s, [0, \infty], L(n_p) \wedge L(n_s))$
	// Recreate minimal time model from updated basic time model.

Table 6: Algorithm 4: Delete Activity

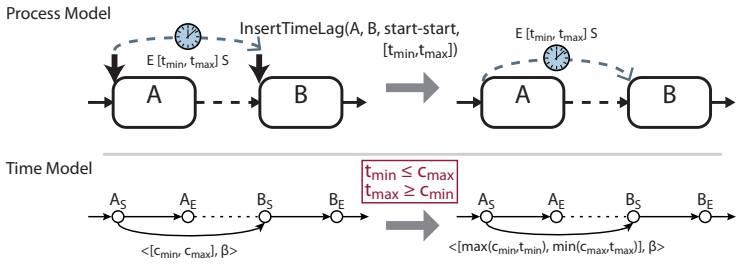


Figure 9: Change Operation: Insert Time Lag

4.2.6. Time Lag Insertion.

Change operation $InsertTimeLag(n_1, n_2, type_{tl}, [t_{min}, t_{max}])$ allows adding a time lag between activities n_1 and n_2. The instants the time lag refers to (i.e., start vs. end) are specified by parameter $type_{tl}$. Adding a time lag is only possible if there exists at least one execution path containing both nodes (i.e., $L(N_{1\langle I_S\rangle}) \wedge L(N_{2\langle I_T\rangle})$ is satisfiable; cf. Table 7) [13].

The two time models are then adapted by adding a constraint $\langle [t_{min}, t_{max}]_{N_{1\langle I_S\rangle} N_{2\langle I_T\rangle}}, \beta\rangle$ between the time-points representing the respective instants (start vs. end) of the two nodes. Basically, this updates each existing implicit constraint $\langle [c_{min}, c_{max}]_{N_{1\langle I_S\rangle} N_{2\langle I_T\rangle}}, \beta\rangle$. Note that this is only possible if the resulting constraint $[\max\{c_{min}, t_{min}\}, \min\{c_{max}, t_{max}\}]$ in the adapted minimal time model still permits at least one value, i.e., allows for at least one possible solution. Accordingly, for the operation to be applicable, for any implicit constraint between $N_{1\langle I_S\rangle}$ and $N_{2\langle I_T\rangle}$ the following must hold: $c_{min} \leq t_{max} \wedge t_{min} \leq c_{max}$.[8] Algorithm 5 defines the respective pre- and post-conditions. After updating the temporal constraints, minimality of the adapted minimal time model must be restored.

4.2.7. Fixed Date Element Insertion.

Inserting a fixed date element (i.e., operation *InsertFDE*) is equivalent to adding a time lag between the special time-point Z (indicating time "0") and the respective instant of the node (cf. Section 3.2). However, in some cases the actual date is not known when inserting the fixed date element. Accordingly, in these cases, no time-specific preconditions have to be observed and no changes to the minimal time model are required (cf. Figure 10). Note that there already exists an implicit constraint between Z and the respective instant of the node. Algorithm 6 (cf. Table 8) defines the respective pre- and post-conditions. After updating the temporal constraints, if the date of the fixed date element is known, minimality of the adapted minimal time model must be restored.

[8] Note that $t_{min} \leq t_{max}$ and $c_{min} \leq c_{max}$ is guaranteed by the time lag and the minimal time model, respectively.

Algorithm 5: InsertTimeLag$(\mathbf{n_1}, \mathbf{n_2}, \mathbf{type_{tl}}, [\mathbf{t_{min}}, \mathbf{t_{max}}])$	
Pre	$\langle I_S \rangle = \begin{cases} S & type_{tl} = \text{start-*} \\ E & type_{tl} = \text{end-*} \end{cases}$, $\langle I_T \rangle = \begin{cases} S & type_{tl} = \text{*-start} \\ E & type_{tl} = \text{*-end} \end{cases}$ $(L(N_{1\langle I_S \rangle}) \wedge L(N_{2\langle I_T \rangle}))$ is satisfiable $\forall \langle [c_{min}, c_{max}]_{N_{1\langle I_S \rangle} N_{2\langle I_T \rangle}}, \beta \rangle \in \mathcal{C} : c_{min} \leq t_{max} \wedge t_{min} \leq c_{max}$
Post	// Update process model: $AddTimeLag(n_1, n_2, \langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle)$
	// Update basic and minimal time model: $AddConstraint(N_{1\langle I_S \rangle}, N_{2\langle I_T \rangle}, [t_{min}, t_{max}], L(N_{1E}) \wedge L(N_{2S}))$
	// Update minimal time model: $\forall \langle [c_{min}, c_{max}]_{N_{1E} N_{2S}}, \beta \rangle \in \mathcal{C}$: $UpdateConstraint(N_{1\langle I_S \rangle}, N_{2\langle I_T \rangle}, [\max\{c_{min}, t_{min}\}, \min\{c_{max}, t_{max}\}], \beta)$

Table 7: Algorithm 5: Insert Time Lag

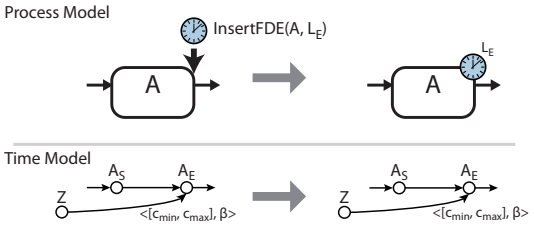


Figure 10: Change Operation: Insert Fixed Date Element

4.2.8. Time Lag and Fixed Date Element Deletion.

Deleting a time lag (operation $DeleteTimeLag(n_1, n_2, type_{tl})$) or fixed date element (operation $DeleteFDE(n, type_{fde})$) is always possible as in both cases no structural restriction is violated and temporal constraints are only removed from the time model. However, as with operation *DeleteActivity* (cf. Table 6), it is not possible to restore minimality of the modified minimal time model. Hence, the minimal time model has to be recalculated from the base time model.

5. Analyzing the Effects of Change Operations on Time-Aware Processes

When changing a time-aware process schema or instance both the corresponding process model and the time models (i.e., basic and minimal time model) must be updated. In this context the minimality of the minimal time model must be restored after each change operation. Only then it becomes possible to ensure that another change within the same transaction may still be applied without violating temporal consistency of the process model. However, calculating the minimal network of a CSTN is expensive regarding computation time. To be more precise its complexity is $O(n^3 2^k)$ with n being the number of time-points and k being the number of observation time-points in the time model. Consequently, there might be significant delays when applying multiple change operations to large time-aware process models. This issue becomes even more pressing in the context of process schema evolution [5] when a potentially large set of process instances shall be dynamically migrated to a new process schema version (i.e., process model). Hence, the maximum effect a particular change has on the minimal time model must be estimated. Based on this, it becomes possible to decide whether or not another change operation may be applied without need to restore minimality of the minimal time model first.

Regarding the change operations introduced in Section 4.2, two different types may be distinguished based on their impact on the time model: 1. Change operations which add a new temporal constraint or further restrict an existing one (cf. change operations *InsertSerial*, *InsertPar*, *InsertCond*, *InsertBranch*, and *InsertTimeLag*) and 2. change operations which relax some temporal constraints by removing others (cf. change operations *DeleteActivity*, *DeleteTimeLag*, and *DeleteFDE*).

Algorithm 6: InsertFDE($\mathbf{n}, \mathbf{type_{fde}}$)	
Pre	$N = \begin{cases} N_S & \text{if } type_{fde} \in \{E_S, L_S\} \\ N_E & \text{if } type_{fde} \in \{E_E, L_E\} \end{cases}$
Init	t = current value of the fixed-date element or $undefined$
	$t_{min} = \begin{cases} 0 & \text{if } t = undefined \\ t & \text{if } type_{fde} \in \{E_S, E_E\} \\ 0 & \text{if } type_{fde} \in \{L_S, L_E\} \end{cases}$
	$t_{max} = \begin{cases} \infty & \text{if } t = undefined \\ \infty & \text{if } type_{fde} \in \{E_S, E_E\} \\ t & \text{if } type_{fde} \in \{L_S, L_E\} \end{cases}$
Post	// Update process model:
	$AddFixedDateElement(n, type_{fde})$
	// Update basic and minimal time model:
	$AddConstraint(Z, N, [0, \infty], L(N))$
	// Update minimal time model:
	$\forall \langle [c_{min}, c_{max}]_{ZN}, \beta \rangle \in \mathcal{C}$:
	$UpdateConstraint(Z, N, [\max\{c_{min}, t_{min}\}, \min\{c_{max}, t_{max}\}], \beta)$

Table 8: Algorithm 6: Insert Fixed Date Element

5.1. Changes Adding or Restricting a Temporal Constraint

Regarding changes making an existing constraint more restrictive, Theorem 1 shows how their maximum effects can be estimated.

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

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

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

To proof Theorem 1 we start by showing how a CSTN can be mapped to a more simple kind of temporal problem, so called *simple temporal networks* [19]:

Definition 5 (Simple Temporal Network)**.** *A* Simple Temporal Network *(STN) is a pair $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of real-valued variables, called* time-points*, and $\mathcal{C}$ is a set of* simple temporal constraints*. Each constraint $c_{XY} \in \mathcal{C}$ has the form $c_{XY} = [x, y]_{XY}$, where X and Y are any time-points in $\mathcal{T}$, and $-\infty \leq x \leq y \leq \infty$ are any real numbers (compare Definition 1).*

A solution *to the STN $(\mathcal{T}, \mathcal{C})$ is a complete set of assignments to the variables in $\mathcal{T}$ that satisfies all of the constraints in $\mathcal{C}$.*

A STN $S = (\mathcal{T}, \mathcal{C})$ is called consistent *if at least one viable solution exists.*

For a STN and respective constraints the following relations can be defined [19]:

Definition 6 (Tighter, Equivalence)**.** *A constraint $c_{XY} = [x_c, y_c]_{XY}$ is* tighter *than constraint $d_{XY} = [x_d, y_d]_{XY}$, denoted as $c_{XY} \subseteq d_{XY}$, if every value allowed by c_{XY} is also allowed by d_{XY}, i.e., $c_{XY} \subseteq d_{XY} \Leftrightarrow x_d \leq x_c \leq y_c \leq y_d$.*

A STN $S = (\mathcal{T}, \mathcal{C}_S)$ is tighter *than STN $T = (\mathcal{T}, \mathcal{C}_T)$, denoted as $S \subseteq T$, if for all corresponding constraints $c_{XY} \in \mathcal{C}_S$ and $d_{XY} \in \mathcal{C}_T$ it holds $c_{XY} \subseteq d_{XY}$. In particular, $\subseteq$ forms a partial order on the set of STNs over a set of time-points $\mathcal{T}$.*

Two STNs S and T are equivalent *—denoted as $S \equiv T$— if they represent the same solution set.*

Based on these notions the minimal network of a STN is defined as follows [19]:

Definition 7 (Minimal Network of a STN)**.** *The* minimal network *of a STN $S = (\mathcal{T}, \mathcal{C})$—denoted as $minimalSTN(S)$—is the unique STN $M = (\mathcal{T}, \mathcal{C}')$ which represents the same solution set as S and where each value allowed by any constraint $c \in \mathcal{C}'$ is part of at least one viable solution of S. More formally, the minimal network is the tightest STN M which is equivalent to S, i.e.,*

$$M = minimalSTN(S) \iff M \equiv S \wedge (\nexists N = (\mathcal{T}, \mathcal{C}_N) : N \subseteq M \wedge N \equiv S)$$

By definition the minimal network for S exists if and only if S is consistent.

Note the similarity between Definition 7 and the one of the minimal network of a CSTN in Definition 4 [18].

We now come back to CSTN and its relationship with STN as discussed in [18]:

Definition 8 (Scenario Projection)**.** *Let $S = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be a CSTN, and s_P be a scenario for the letters in P (cf. Definition 2). The* projection *of S onto the scenario s_P—denoted by $scProj(S, s_P)$—is a STN $Q = (\mathcal{T}_s^+, \mathcal{C}_s^+)$, where:*

- $\mathcal{T}_s^+ = \{T \in \mathcal{T} : s_P(L(T)) = true\}$*; and*
- $\mathcal{C}_s^+ = \{[x, y]_{XY} \mid$ *for some* $\beta, \langle [x, y]_{XY}, \beta \rangle \in \mathcal{C}$ *and* $s_P(\beta) = true\}$

Based on this we can define the following corollary to Definition 3:

Corollary 1 (Weak Consistency)**.** *A CSTN $S = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ is weakly consistent iff for each scenario s_P over P the respective scenario projection $scProj(S, s_P)$ is consistent.*

Thus, for the minimal network of a CSTN we can define the following lemma based on the minimal network of STN:

Lemma 1 (Minimal Network)**.** *Let $minimalCSTN(S)$ denote the minimal network of CSTN S (cf. Definition 4) and $minimalSTN(T)$ the minimal network of STN T (cf. Definition 7). Then, it holds that:*

$$\forall_{s_P} : scProj(minimalCSTN(S), s_P) \equiv minimalSTN(scProj(S, s_P))$$

Proof. This follows directly from Definition 4 and Corollary 1. Particularly, according to Definition 4 $minimalCSTN(S)$ must have the same set of solutions as S which, in turn, are given by $minimalSTN(scProj(S, s_P))$. □

We will now show that Theorem 1 holds for STN. But first we need to introduce some additional concepts and properties of STNs.

Definition 9 (Inverse, Composition, Intersection)**.** *For a constraint $c_{XY} = [x, y]_{XY}$ the* inverse *constraint is*

$$c_{YX} = -c_{XY} = [-y, -x]_{YX} .$$

The composition *of two constraint $c_{XY} = [x_c, y_c]_{XY}$ and $d_{YZ} = [x_d, y_d]_{YZ}$ is defined as*

$$c_{XY} \oplus d_{YZ} = [x_c + x_d, y_c + y_d]_{XZ} .$$

The intersection *of two constraint $c_{XY} = [x_c, y_c]_{XY}$ and $d_{XY} = [x_d, y_d]_{XY}$ is defined as*

$$c_{XY} \cap d_{XY} = \begin{cases} [\max\{x_c, x_d\}, \min\{y_c, y_d\}]_{XY} & \text{if } \max\{x_c, x_d\} \leq \min\{y_c, y_d\} \\ \emptyset & \text{else} \end{cases} .$$

Table 9 details useful properties of the operations defined by Definition 6 and Definition 9.

A *path* in an STN is a sequence of time-points which does not contain any subsequent pair of time-points twice, formally:

Operator Precedence	$\oplus \rightarrow \cap$
Associativity	$(c_{XY} \cap d_{XY}) \cap e_{XY} = c_{XY} \cap (d_{XY} \cap e_{XY})$ $(c_{WX} \oplus d_{XY}) \oplus e_{YZ} = c_{WX} \oplus (c_{XY} \oplus e_{YZ})$
Commutativity	$c_{XY} \cap d_{XY} = d_{XY} \cap c_{XY}$
Distributivity	$c_{XY} \oplus (c_{YZ} \cap d_{YZ}) = (c_{XY} \oplus c_{YZ}) \cap (c_{XY} \oplus d_{YZ})$
Negation	$-(-c_{XY}) = -c_{YX} = c_{XY}$ $-(c_{XY} \oplus c_{YZ}) = (-c_{ZY}) \oplus (-c_{YX})$
Reflexivity	$c_{XY} \subseteq c_{XY}$
Transitivity	$(c_{XY} \subseteq d_{XY} \wedge d_{XY} \subseteq e_{XY}) \Rightarrow c_{XY} \subseteq e_{XY}$

Table 9: Properties of Composition, Intersection and Tighter

Definition 10 (Path). *Let $S = (\mathcal{T}, \mathcal{C})$ be a STN. Then for $n \geq 3$ a* path *p from X to Y is a sequence of time-points $p = \langle K_1, \ldots, K_n \rangle; K_1, \ldots, K_n \in \mathcal{T}$ with $K_1 = X$, $K_n = Y$ and $\forall i \in 1, \ldots, n-1 : \nexists j \in 1, \ldots, n-1 : (K_i = K_j \wedge K_{i+1} = K_{j+1}) \vee (K_i = K_{j+1} \wedge K_{i+1} = K_j)$, i.e., a path does not contain any cycles.*

The constraint c_p of path $p = \langle K_1, \ldots, K_n \rangle$ on constraint set $\mathcal{C}$ is given by $c_p = c_{K_1K_2} \oplus c_{K_2K_3} \oplus \ldots \oplus c_{K_{n-2}K_{n-1}} \oplus c_{K_{n-1}K_n}; c_{K_1K_2} \ldots c_{K_{n-1}K_n} \in \mathcal{C}$.

Furthermore, $\mathcal{P}_{XY} \subseteq 2^{\mathcal{T}}$ denotes the set of all paths from X to Y in $\mathcal{T}$.

These definitions enable us to provide a more productive characterization of a minimal STN [19, 20]:

Lemma 2 (Minimal STN). *A STN $S = (\mathcal{T}, \mathcal{C})$ is* minimal *iff for each constraint $c_{XY} \in \mathcal{C}$ it holds that c_{XY} is tighter than the constraint c_p of any path $p = \langle X, Z, Y \rangle$ of length three from X to Y. Formally:*

$$S \text{ is minimal} \Leftrightarrow \forall c_{XY} \in \mathcal{C}, \forall Z \in \mathcal{T}, p = \langle X, Z, Y \rangle \in \mathcal{P}_{XY} : c_{XY} \subseteq c_p = c_{XZ} \oplus c_{ZY}$$

Proof. See proof of Lemma 5.10 in [19]. In particular, it is easy to verify that if $\exists c_{XY} \in \mathcal{C}, \exists Z \in \mathcal{T} : \exists x \in c_{XY} : x \notin c_{XZ} \oplus c_{ZY}$ then x cannot be part of any solution of S and thus S cannot be minimal. □

This gives rise to the following two corollaries, which show how to calculate the minimal STN M of STN S:

Corollary 2 (Minimal STN). *The minimal STN $M = (\mathcal{T}, \mathcal{C}')$ for STN $S = (\mathcal{T}, \mathcal{C})$ can be calculated as follows:*

Let $N^1 = (\mathcal{T}, \mathcal{C}^1) = S$ and let $N^{k+1} = (\mathcal{T}, \mathcal{C}^{k+1})$ with

$$c_{XY}^{k+1} = c_{XY} \cap \bigcap_{Z \in \mathcal{T}} c_{XZ}^k \oplus c_{ZY}^k$$

where $c_{XY} \in \mathcal{C}$, $c_{XZ}^k, c_{ZY}^k \in \mathcal{C}^k$ and $c_{XY}^{n+1} \in \mathcal{C}^{k+1}$. Then, either $\exists n \geq 1$ such that $N^n = N^{n+1} = M$ is the minimal STN or $\exists n \geq 1$ such that $\exists c_{XY}^n \in \mathcal{C}^n : c_{XY}^n = \emptyset$ in which case STN S is not consistent (i.e., no minimal network exists).

Proof. This follows directly from the fact that the minimal network of an STN can be derived by enforcing 3-path consistency [19]. Also compare the PC1-Algorithm for calculating the minimal network presented in [19], which is shown to be correct as well as complete. In particular, if $N^n = N^{n+1}$ it is easy to see that $\forall c_{XY}^n, c_{XZ}^n, c_{ZY}^n \in \mathcal{C}^n : c_{XY}^n \subseteq c_{XZ}^n \oplus c_{ZY}^n$ and hence N^n is minimal. Furthermore, for each c_{XY}^{k+1} it holds that $c_{XY}^k \supseteq c_{XY}^{k+1}$ and because in each step $k \rightarrow k+1$ at least one c_{XY}^{k+1} is modified (i.e., $c_{XY}^k \supset c_{XY}^{k+1}$) for $k \rightarrow \infty$ it follows that either $\exists n \geq 1 : \forall k \geq n : \forall c_{XY}^k \in C^k, \forall c_{XY}^{k+1} \in C^{k+1} : c_{XY}^k = c_{XY}^{k+1}$ or $\exists n \geq 1 : \exists c_{XY}^n \in C^n : c_{XY}^n = \emptyset$, i.e., a fix point exists. □

Corollary 3 (Minimal STN)**.** *The minimal STN $M = (\mathcal{T}, \mathcal{C}')$ for STN $S = (\mathcal{T}, \mathcal{C})$ is given by $c'_{XY} \in \mathcal{C}'$ with:*

$$\begin{aligned} c'_{XY} &= c_{XY} \cap \bigcap_{K_1 \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1Y}) \cap \bigcap_{K_1,K_2 \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1K_2} \oplus c_{K_2Y}) \cap \dots \\ &\quad \cap \bigcap_{K_1,\dots,K_n \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1K_2} \oplus \dots \oplus c_{K_{n-1}K_n} \oplus c_{K_nY}) \\ &= c_{XY} \cap \bigcap_{p \in \mathcal{P}_{XY}} c_p \end{aligned}$$

where $n = |T| - 2$ and $c_{ij} \in \mathcal{C}$.

In particular, note that without loss of generality we assume that $\forall c_{XY} \in \mathcal{C} : c_{YX} = -c_{XY} \in \mathcal{C}$ (cf. Definition 9).

Proof. Based on Corollary 2 we know that for $M = N^{n+1}$ it holds that

$$c^{n+1}_{XY} = c_{XY} \cap \bigcap_{K_1 \in \mathcal{T}} \left(c^n_{XK_1} \oplus c^n_{K_1Y}\right) \tag{1}$$

furthermore we know that for any $N^i, 2 \leq i \leq n$

$$c^i_{XK_1} = c_{XK_1} \cap \bigcap_{K_2 \in \mathcal{T}} \left(c^{i-1}_{XK_2} \oplus c^{i-1}_{K_2K_1}\right) \tag{2}$$

and similarly for $c^i_{K_1Y}$. Applying equation 2 to equation 1 we obtain

$$\begin{aligned} & c^{n+1}_{XY} \\ &= c_{XY} \cap \bigcap_{K_1 \in \mathcal{T}} \left(\left(c_{XK_1} \cap \bigcap_{K_2 \in \mathcal{T}} \left(c^{n-1}_{XK_2} \oplus c^{n-1}_{K_2K_1}\right) \right) \right. \\ &\qquad \left. \oplus \left(c_{K_1Y} \cap \bigcap_{K_2 \in \mathcal{T}} \left(c^{n-1}_{K_1K_2} \oplus c^{n-1}_{K_2Y}\right) \right) \right) \\ &= c_{XY} \cap \bigcap_{K_1 \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1Y}) \cap \bigcap_{K_1,K_2 \in \mathcal{T}} \left(c_{XK_1} \oplus c^{n-1}_{K_1K_2} \oplus c^{n-1}_{K_2Y}\right) \\ &\quad \cap \bigcap_{K_1,K_2 \in \mathcal{T}} \left(c^{n-1}_{XK_2} \oplus c^{n-1}_{K_2K_1} \oplus c_{K_1Y}\right) \\ &\quad \cap \bigcap_{K_1,K_2 \in \mathcal{T}} \left(c^{n-1}_{XK_2} \oplus c^{n-1}_{K_2K_1} \oplus c^{n-1}_{K_1K_2} \oplus c^{n-1}_{K_2Y}\right) \\ &= c_{XY} \cap \bigcap_{K_1 \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1Y}) \cap \bigcap_{K_1,K_2 \in \mathcal{T}} \left(c_{XK_1} \oplus c^{n-1}_{K_1K_2} \oplus c^{n-1}_{K_2Y}\right) \end{aligned}$$

If we repeat the same steps for $c^{n-1}_{ij}, \dots, c^2_{ij}$ and note that $c^1_{ij} = c_{ij}$, we obtain

$$\begin{aligned} c^{n+1}_{XY} &= c_{XY} \cap \bigcap_{K_1 \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1Y}) \cap \bigcap_{K_1,K_2 \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1K_2} \oplus c_{K_2Y}) \cap \dots \\ &\quad \cap \bigcap_{K_1,\dots,K_n \in \mathcal{T}} (c_{XK_1} \oplus c_{K_1K_2} \oplus \dots \oplus c_{K_{n-1}K_n} \oplus c_{K_nY}) \end{aligned}$$

which completes the proof. □

This finally brings us to Theorem 2 which is the equivalence of Theorem 1 for STN.

Theorem 2 (Restricting a constraint in a minimal STN)**.** *Let $M = (\mathcal{T}, \mathcal{C}_M)$ be the minimal network for STN $S = (\mathcal{T}, \mathcal{C}_S)$ (i.e., minimalSTN$(S) = M$). Further, let $M^* = (\mathcal{T}, \mathcal{C}_{M^*})$ be the STN derived from M by replacing constraint $c_{AB} = [x, y]_{AB} \in \mathcal{C}_M$ with $c^*_{AB} = [x + \sigma, y - \rho]_{AB}$; $\sigma, \rho \geq 0$ (i.e., $c^*_{AB} \subseteq c_{AB}$ and $C^*_M = C_M \setminus c_{AB} \cup \{c^*_{AB}\}$).*

Then for the minimal network $N = (\mathcal{T}, \mathcal{C}_N) =$ minimalSTN(M^*) of M^* it holds that for any constraint $c'_{XY} = [x', y']_{XY} \in \mathcal{C}_N$ the lower bound is increased by at most $\delta = \max\{\rho, \sigma\}$ and the upper bound is decreased by at most δ compare to the corresponding constraint $c_{XY} = [x, y]_{XY} \in \mathcal{C}_M$. Formally,

$$\begin{aligned} &\forall\, [x, y]_{XY} \in \mathcal{C}_M, [x', y']_{XY} \in \mathcal{C}_N : \\ &\qquad [x, y]_{XY} \supseteq [x', y']_{XY} \supseteq [x + \delta, y - \delta]_{XY} \end{aligned}$$

Proof. The first part, i.e., $[x, y]_{XY} \supseteq [x', y']_{XY}$ follows directly from the definition of the minimal network (cf. Definition 7).

For the second part, i.e., $[x', y']_{XY} \supseteq [x + \delta, y - \delta]_{XY}$, remember that according to Corollary 3 it holds, that for any STN $M = (\mathcal{T}, \mathcal{C}_M)$ the minimal network $N = (\mathcal{T}, \mathcal{C}_N)$ can be calculated based on the intersection of all paths between two time-points. Formally:

$$\forall c'_{XY} \in \mathcal{C}_N, c_{XY} \in \mathcal{C}_M : c'_{XY} = c_{XY} \cap \bigcap_{p \in \mathcal{P}_{XY}} c_p \tag{3}$$

where $p \in \mathcal{P}_{XY}$ is a path $p = \langle X, K_1, \ldots, K_n, Y \rangle$ from X to Y, c_p is the corresponding constraint in $\mathcal{C}_M$, and $c'_{XY} \in \mathcal{C}_N$.

Furthermore, for the constraint c_p of any path $p = \langle K_1, \ldots, K_n \rangle$ on constraint set $\mathcal{C}_M$ with constraints $c_{K_1K_2}, \ldots, c_{K_{n-1}K_n} \in \mathcal{C}_M$ it holds that

$$\begin{aligned} c_p = (c_{K_1K_2} \oplus \ldots \oplus c_{K_{n-1}K_n}) &= \left[\sum_{l=2,\ldots,n} x_{K_{l-1}K_l}, \sum_{l=2,\ldots,n} y_{K_{l-1}K_l} \right]_{K_1K_n} \\ &= [x_p, y_p]_{K_1K_n} \end{aligned} \tag{4}$$

and for $\bigcap_{p \in \mathcal{P}_{XY}} c_p$ it holds (cf. Definition 9)

$$\bigcap_{p \in \mathcal{P}_{XY}} c_p = \bigcap_{p \in \mathcal{P}_{XY}} [x_p, y_p]_p = \left[\max_{p \in \mathcal{P}_{XY}} \{x_p\}, \min_{p \in \mathcal{P}_{XY}} \{y_p\} \right]_{XY}$$

Now, if constraint $c_{AB} = [x_{AB}, y_{AB}]_{AB} \in \mathcal{C}_M$ is restricted to $c^*_{AB} = [x_{AB} + \sigma, y_{AB} - \rho]_{AB}$ in $\mathcal{C}_{M^*}$ the constraint c_p of any path $p = \langle K_1, \ldots, K_i, A, B, K_{i+3}, \ldots, K_n \rangle \in \mathcal{P}_{XY}$ containing A, B in equation 3 will be replaced by c^*_p, where according to equation 4 for the constraint $c^*_p = \left[x^*_p, y^*_p\right]_{XY}$ of path p on constraint set C_{M^*} the following holds

$$\begin{aligned} x^*_p &= \left(\sum_{l=2,\ldots,i} x_{K_{l-1}K_l} \right) + x_{K_iA} + x^*_{AB} + x_{BK_j} + \left(\sum_{l=j+1,\ldots,n} x_{K_{l-1}K_l} \right) \\ &= \left(\sum_{l=2,\ldots,n} x_{K_{l-1}K_l} \right) + \sigma && x^*_{AB} = x_{AB} + \sigma \\ &= x_p + \sigma \end{aligned}$$

and correspondingly

$$\begin{aligned} y^*_p &= \left(\sum_{l=2,\ldots,i} y_{K_{l-1}K_l} \right) + y_{K_iA} + y^*_{AB} + y_{BK_j} + \left(\sum_{l=j+1,\ldots,n} y_{K_{l-1}K_l} \right) \\ &= y_p - \rho \end{aligned}$$

Consequently, it holds:

$$c_p^* = \left[x_p^*, y_p^*\right]_{XY} = [x_p + \sigma, y_p - \rho]_{XY} \tag{5}$$

i.e., the lower bound of the constraint of any path p containing A, B is increased by σ and the upper bound is decreased by ρ. With similar argument we can show that for any path $p = \langle K_1, \ldots, B, A, \ldots, K_n\rangle$ containing the inverse c_{BA} of c_{AB} the lower bound of the constraint is increased by ρ and the upper bound is decreased by σ, i.e., $\left[x_p^*, y_p^*\right]_{XY} = [x_p + \rho, y_p - \sigma]_{XY}$. Any path not containing c_{AB} (or c_{BA}) remains unchanged. Hence for any path $p \in \mathcal{P}_{xy}$ we know that

$$[x_p, y_p]_{XY} \supseteq \left[x_p^*, y_p^*\right]_{XY} \supseteq [x_p + \max\{\sigma, \rho\}, y_p - \max\{\sigma, \rho\}]_{XY} = [x_p + \delta, y_p - \delta]_{XY} \tag{6}$$

with $\delta = \max\{\sigma, \rho\}$.

Due to M being minimal (cf. Theorem 2) $c_{XY} = c_{XY} \cap \bigcap_{p\in\mathcal{P}_{XY}} c_p$ must hold for any path in $\mathcal{C}_M$ (including the ones containing c_{AB}). Furthermore, $c'_{XY} = c_{XY} \cap \bigcap_{p\in\mathcal{P}_{XY}} c_p^*$ and $\forall p \in \mathcal{P}_{XY} : c_p^* \subseteq c_p$ hold for any path in C_{M^*}. Hence, with $c_{XY} = [x_c, y_c]_{XY}$ it follows:

$$\begin{aligned}
c'_{XY} =& c_{XY} \cap \bigcap_{p\in\mathcal{P}_{XY}} c_p^* = \left(c_{XY} \cap \bigcap_{p\in\mathcal{P}_{XY}} c_p\right) \cap \bigcap_{p\in\mathcal{P}_{XY}} c_p^* && \text{with } c_{XY}=c_{XY}\cap\textstyle\bigcap_{p\in\mathcal{P}_{XY}} c_p \\
=& c_{XY} \cap \left(\bigcap_{p\in\mathcal{P}_{XY}} c_p \cap \bigcap_{p\in\mathcal{P}_{XY}} c_p^*\right) \\
=& c_{XY} \cap \left[\max\left\{\max_{p\in\mathcal{P}_{XY}}\{x_p\}, \max_{p\in\mathcal{P}_{XY}}\{x_p^*\}\right\},\right. \\
& \left.\min\left\{\min_{p\in\mathcal{P}_{XY}}\{y_p\}, \min_{p\in\mathcal{P}_{XY}}\{y_p^*\}\right\}\right]_{XY} && \text{cf. Definition 9 and } c_p = \left[x_p, y_p\right]_p;\ c_p^* = \left[x_p^*, y_p^*\right]_p \\
\supseteq& c_{XY} \cap \left[\max_{p\in\mathcal{P}_{XY}}\{x_p + \delta\}, \min_{p\in\mathcal{P}_{XY}}\{x_p - \delta\}\right]_{XY} && \text{cf. Equation 6} \\
=& c_{XY} \cap \left[\max_{p\in\mathcal{P}_{XY}}\{x_p\} + \delta, \min_{p\in\mathcal{P}_{XY}}\{x_p\} - \delta\right]_{XY} \\
\supseteq& c_{XY} \cap [x_c + \delta, y_c - \delta]_{XY} && M \text{ minimal, i.e., } \max_{p\in\mathcal{P}_{XY}}\{x_p\}\leq x_c \text{ and } \min_{p\in\mathcal{P}_{XY}}\{y_p\}\geq y_c \\
=& [x_c + \delta, y_c - \delta]_{XY} && c_{XY}=[x_c,y_c]_{XY}
\end{aligned}$$

which completes the proof. □

We know show, that since Theorem 2 holds for STN, Theorem 1 must hold for the minimal network of a weakly consistent CSTN.

Proof (Theorem 1). Based on the definition of the minimal network for CSTN we know that for the constraints $\mathcal{C}_N$ of the minimal network $N = \mathit{minimalCSTN}(M^*)$ of CSTN M^* it holds that

$$\forall c_{XY} = \langle [x, y]_{XY}, \gamma\rangle \in \mathcal{C}_{M^*}, c'_{XY} = \langle [x', y']_{XY}, \gamma\rangle \in \mathcal{C}_N : [x, y] \supseteq [x', y']$$

Let $\mathit{restrictSTN}_\delta(S) = (\mathcal{T}, \mathcal{C}^-)$ denote the STN derived from STN $S = (\mathcal{T}, \mathcal{C})$ where for each constraint in C the bounds are restricted by δ, i.e., the upper bound is decreased by δ and the lower bound is increased by δ:

$$\forall\, [x, y]_{XY} \in C : \left[x^-, y^-\right]_{XY} \in \mathcal{C}^- \wedge (x^- = x - \delta) \wedge (y^- = y - \delta)$$

Likewise, let $\mathit{restrictCSTN}_\delta(T) = \langle \mathcal{T}, \mathcal{C}^-, L, \mathcal{OT}, \mathcal{O}, P\rangle$ denote the CSTN derived from CSTN $T = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P\rangle$ where for each constraint in C the bounds are restricted by δ. Then it can be easily verified that for CSTN T the following holds:

$$\mathit{scProj}(\mathit{restrictCSTN}_\delta(T), s_P) \equiv \mathit{restrictSTN}_\delta(\mathit{scProj}(T, s_P))$$

In particular, from Definition 8 it follows that $scProj(\cdot, s_P)$ does not alter the values of the involved constraints.

Hence, we can conclude that for each scenario it follows:

$$\begin{aligned}\forall_{s_P} : scProj(N, s_P) &\supseteq minimalSTN(scProj(M^*, s_P)) \\ &\supseteq restrictSTN_\delta(scProj(M, s_P)) \qquad \text{(cf. Theorem 2)} \\ &= scProj(restrictCSTN_\delta(M), s_P)\end{aligned}$$

As $scProj(\cdot, s_P)$ does not alter the values of the involved constraints (cf. Definition 2) it follows that

$$N \supseteq restrictSTN_\delta(M)$$

and thus it holds:

$$\forall\langle[x', y']_{XY}, \gamma\rangle \in \mathcal{C}_N, \forall\langle[x, y]_{XY}, \gamma\rangle \in M : \langle[x', y']_{XY}, \gamma\rangle \supseteq \langle[x + \delta, y - \delta]_{XY}, \gamma\rangle$$

which completes the proof. □

Finally, note that with similar argument this can also be shown for other consistency notions of CSTN, i.e., strong consistency and dynamic consistency (cf. [8]). Particularly, in both cases a corresponding STN representation exists [8]. But this is out of scope of this paper.

To illustrate Theorem 1, assume that due to a change operation a constraint $[x, y]_{XY}$ in the minimal time model is restricted to $[x^*, y^*]_{XY} = [x + \rho, y - \sigma]_{XY}$ and afterwards minimality of the time model is restored. Theorem 1 now states that any constraint $[u, v]_{UV}$ in the original minimal time model is restricted to at most $[u', v']_{UV} = [u + \delta, v - \delta]_{UV}$; $\delta = \max\{\rho, \sigma\}$ in the new minimal time model.

Reconsider change operation *InsertSerial* (cf. Table 3). Assume that the minimal time model is adapted as described by Algorithm 1. The next step would be to restore minimality of this time model. First, note that the constraints introduced by the newly added control edges as well as activity do not affect the other constraints when restoring minimality. By construction, their effects are already incorporated in the constraint between time-points N_{1E} and N_{2S}, which is updated in the context of the operation (cf. Algorithm 1). In particular, the added constraints are nested and consistent with the constraint between N_{1E} and N_{2S} (see [21] for details). The only change having an effect on the resulting minimal time model is the one restricting constraint $[c_{min}, c_{max}]$ between N_{1E} and N_{2S} to $[\max\{c_{min}, d_{min}\}, c_{max}]$. Note that if the constraint is not changed (i.e., $d_{min} \leq c_{min}$), the existing constraints of the minimal time model also need not be changed. Otherwise, the lower bound of the constraint is increased by $\delta = d_{min} - c_{min}$. Theorem 1 implies that the upper and lower bound of any other constraint in the new minimal time model will be restricted by at most δ as well. Thus we are able to approximate the maximum difference between the new minimal time model and the original one.

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

Similar rules apply for all other change operations adding or restricting a temporal constraint. For change operation *InsertPar* (cf. Algorithm 2), in particular, the change relevant to the minimal time model is the one restricting the constraint between time-points N_{pE} and N_{sS} to $[\max\{c_{min}, d_{min}\}, c_{max}]_{N_{pE}N_{sS}}$, i.e., its impact is at most $\delta = \max\{0, d_{min} - c_{min}\}$. Similar, for *InsertCond* (cf. Algorithm 3) the change relevant to the minimal time model is the one restricting the constraint between time-points N_{1E} and N_{2S} to $[\max\{c_{min}, d_{min}\}, c_{max}]$, i.e., the impact on the other constraints is at most $\delta = \max\{0, d_{min} - c_{min}\}$. Finally, for *InsertTimeLag* and *InsertFDE* the maximum impact corresponds to $\delta = \max\{0, t_{min} - c_{min}, t_{max} - c_{max}\}$ (cf. Algorithm 5 and 6).

5.2. Changes Relaxing or Removing an Existing Constraint

When removing an existing explicit constraint from the time model this basically results in the possible relaxation of some implicit constraints. As discussed in Section 4.2 it is not possible to restore minimality of a modified minimal time model after relaxing one of its constraints. This is due to the fact, that it is not easily possible to determine which other constraints have to be relaxed and to what extend.

However, relaxing a constraint in the minimal time model only results in the relaxation of other constraints. Particularly, no existing constraint in the minimal time model is restricted by this change, i.e., the impact is at most $\delta = 0$ (i.e., $\delta \leq 0$). Thus, it is not necessary to restore minimality of the minimal time model after each change operation. Instead it is sufficient to only restore minimality of the minimal time model in case the precondition of a subsequent change operations cannot be met. Particularly, in such a case it is necessary to check whether the change operation indeed violates temporal consistency of the process model or whether the current approximation of the minimal time model is too strict.

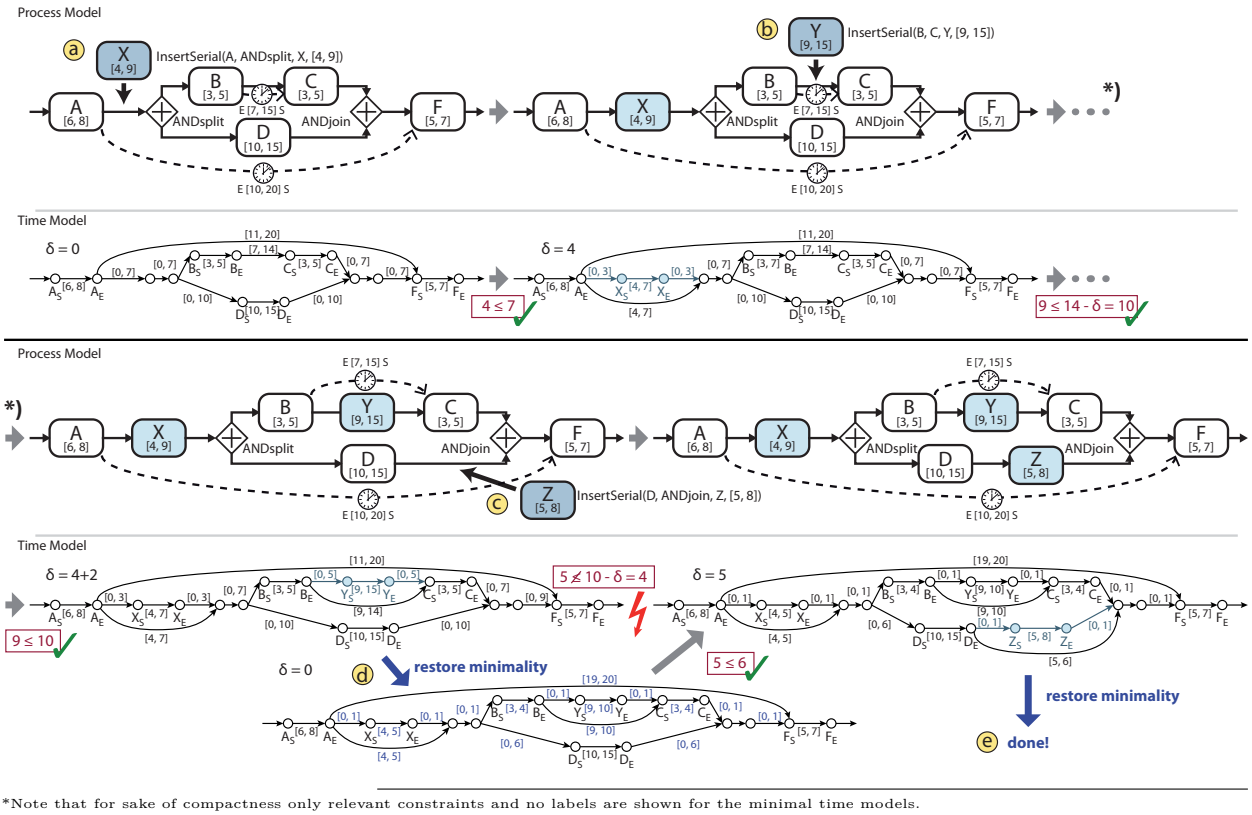
5.3. Applying Multiple Change Operations

Based on these observations it becomes possible to apply a sequence of change operations to a process model within a single transaction (e.g., to insert and/or delete multiple activities) without need to restore minimality of the minimal time model after each change. Particularly, in case a sequence of change operations $op_1, \ldots, op_n$ with impacts $\delta_1, \ldots, \delta_n$ shall be applied to a process model, it will be sufficient to consider the aggregated impacts of the previously applied change operations. Practically, for operation op_i approximated constraint $[x + \sum_{j=1}^{i-1} \delta_j, y - \sum_{j=1}^{i-1} \delta_j]_{XY}$ needs to be considered to determine whether the change operation may be applied. Note that this will significantly reduce complexity when applying multiple change operations. However, the actual savings depend on the strictness of the constraints of the time-aware process model; if the latter is "heavily" constrained, only few change operations can be applied without need to restore minimality of the minimal time model. In turn, if the constraints are "weak" multiple change operations may be applied at once, without having to restore minimality of the minimal time model between changes.

We illustrate our approach along the example from Figure 11. It depicts a process model and corresponding minimal time model to which a series of three change operations ⓐ-ⓒ shall be applied. First, `X` having duration $[4, 9]$ shall be inserted between `A` and `ANDsplit` (Figure 11 ⓐ). This is possible without violating temporal consistency of the process model since the minimum duration of `X` is lower than the maximum time distance between `A` and `ANDsplit` (i.e., $4 \leq 7$). After performing the change, the value used for approximating the minimal time model becomes $\delta = 4 - 0 = 4$. Next, `Y` shall be inserted between `B` and `C` (Figure 11 ⓑ). Again this is possible since the minimum duration is lower than the approximated maximum time distance (i.e., $9 \leq 14 - \delta = 10$). Afterwards δ is increased to $\delta = 4 + (9 - 7) = 6$. Subsequently, inserting `Z` with duration $[5, 8]$ between `D` and `ANDjoin` (Figure 11 ⓒ) is not possible based on the approximated minimal time model as the precondition of the respective change operation cannot be met (i.e., $5 \not\leq 10 - \delta = 4$). Hence, minimality of the minimal time model must be restored (Figure 11 ⓓ). Afterwards, inserting `Z` becomes possible as for the new minimal time model the precondition of the operation is met. Finally, minimality of the last minimal time model must be restored (Figure 11 ⓔ).

6. Evaluation

The presented approach was implemented as a proof-of-concept prototype based on the AristaFlow BPM Suite [5]. This prototype enables users to create time-aware process models and to automatically generate respective time models based on CSTN (cf. Figure 12). Further, the presented change operations may be applied to both process models and corresponding instances. Overall, the prototype demonstrates the applicability of our approach. The screenshot from Figure 12 shows four windows: at the top, a process model from the healthcare domain comprising several temporal constraints is shown. At the bottom left, the automatically generate base time model is depicted. At the bottom right, the corresponding minimal time model is shown. Finally, the right side displays the available set of change operations. Whether a particular change operation may be applied is decided by checking both structural and temporal preconditions. When applying the operation to the process model (i.e., schema or instance) all three models are updated



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

Figure 11: Applying Multiple Change Operations to a Process Model

simultaneously as described in Section 4.1. Altogether the prototype allows us to efficiently provide the required flexibility for time-aware processes.

7. Conclusion

Time constitutes a fundamental concept regarding the operational support of business processes in a PAIS. In business, where missed deadlines and violations of temporal constraints might cause significant problems, it is crucial for enterprises to be able to efficiently control and monitor these temporal constraints during run-time. Since process execution does not always stick to the plan in practice, enterprises must be further able to flexibly react to deviations in a time-aware process without affecting other properties of the process. This paper considered dynamic process changes in the context of time-aware processes. First, we defined change operations for time-aware processes. Second, we specified pre- and post-conditions for these operations, which ensure that changed process models remain temporally consistent. Third, we analyzed the effects respective change operations have on the temporal constraints of the process model. Fourth, we approximated the resulting temporal properties of the entire process model. In particular, this allows us to significantly reduce the complexity of the required time calculations in the context of subsequent changes. In order to demonstrate the feasibility of the presented approach, a powerful proof-of-concept prototype was implemented.

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

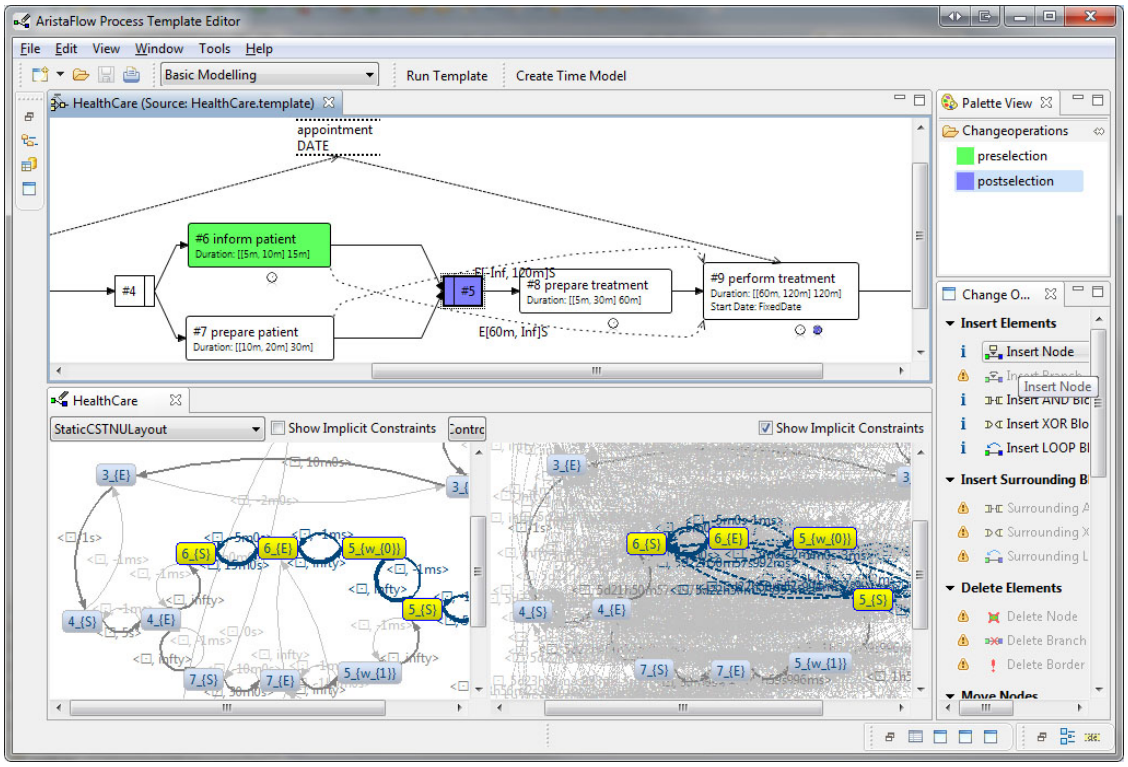


Figure 12: Screenshot of the Prototype (based on the AristaFlow BPM Suite)

References

[1] A. Lanz, B. Weber, M. Reichert, Time patterns for process-aware information systems, Requirements Engineering(online first). doi:10.1007/s00766-012-0162-3.

[2] C. Combi, M. Gozzi, R. Posenato, G. Pozzi, Conceptual modeling of flexible temporal workflows, ACM Transactions on Autonomous and Adaptive Systems 7 (2) (2012) 19:1–19:29. doi:10.1145/2240166.2240169.

[3] J. Eder, P. Euthimios, H. Pozewaunig, M. Rabinovich, Time management in workflow systems, in: Proceedings of the 3rd International Conference on Business Information Systems (BIS'99), Springer Berlin / Heidelberg, 1999, pp. 265–280.

[4] A. Lanz, R. Posenato, C. Combi, M. Reichert, Controllability of time-aware processes at run time, in: Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13), no. 8185 in Lecture Notes in Computer Science, Springer, 2013, pp. 39–56. doi:10.1007/978-3-642-41030-7_4.

[5] M. Reichert, B. Weber, Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies, Springer Berlin / Heidelberg, 2012. doi:10.1007/978-3-642-30409-5.

[6] C. Bettini, X. S. Wang, S. Jajodia, Temporal reasoning in workflow systems, Distributed and Parallel Databases 11 (3) (2002) 269–306. doi:10.1023/A:1014048800604.

[7] J. Eder, W. Gruber, E. Panagos, Temporal modeling of workflows with conditional execution paths, in: M. T. Ibrahim, J. Küng, N. Revell (Eds.), Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA'00), Vol. 1873 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2000, pp. 243–253.

[8] I. Tsamardinos, T. Vidal, M. Pollack, CTP: A new constraint-based formalism for conditional, temporal planning, Constraints 8 (4) (2003) 365–388. doi:10.1023/A:1025894003623.

[9] S. W. Sadiq, O. Marjanovic, M. E. Orlowska, Managing change and time in dynamic workflow processes, International Journal of Cooperative Information Systems 9 (1-2) (2000) 93–116. doi:10.1142/S0218843000000077.

[10] S. Rinderle, M. Reichert, P. Dadam, Correctness criteria for dynamic changes in workflow systems: A survey, Data & Knowledge Engineering 50 (1) (2004) 9–34.

[11] S. Rinderle-Ma, M. Reichert, B. Weber, On the formal semantics of change patterns in process-aware information systems, in: Q. Li, S. Spaccapietra, E. Yu, A. Olivé (Eds.), Proceedings of the 27th International Conference on Conceptual Modeling (ER'08), Vol. 5231 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 279–293. doi:10.1007/978-3-540-87877-3_21.

[12] O. Marjanovic, M. E. Orlowska, On modeling and verification of temporal constraints in production workflows, Knowledge and Information Systems 1 (2) (1999) 157–192.

[13] A. Lanz, M. Reichert, B. Weber, A formal semantics of time patterns for process-aware information systems, Tech. Rep. UIB-2013-02, University of Ulm (2013).
URL dbis.eprints.uni-ulm.de/894/

[14] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change support features - enhancing flexibility in process-aware information systems, Data & Knowledge Engineering 66 (3) (2008) 438–466. doi:10.1016/j.datak.2008.05.001.
[15] J. Vanhatalo, H. Völzer, F. Leymann, Faster and more focused control-flow analysis for business process models through sese decomposition, in: B. Krämer, K.-J. Lin, P. Narasimhan (Eds.), Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07), Vol. 4749 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 43–55. doi:10.1007/978-3-540-74974-5_4.
[16] M. Reichert, S. Rinderle, U. Kreher, P. Dadam, Adaptive process management with ADEPT2, in: Proceedings of the International Conference on Data Engineering (ICDE'05), IEEE Computer Society Press, 2005, pp. 1113–1114.
[17] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51. doi:10.1023/A:1022883727209.
[18] L. Hunsberger, R. Posenato, C. Combi, The dynamic controllability of conditional STNs with uncertainty, in: Proceedings of the Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx), 2012.
[19] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95. doi:http://dx.doi.org/10.1016/0004-3702(91)90006-6.
[20] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[21] J. Chen, Y. Yang, Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems, ACM Transactions on Software Engineering and Methodology 20 (3) (2011) 9:1–9:23.