Controlling Time-Awareness in Modularized Processes (Extended Version)

Andreas Lanz[a], Roberto Posenato[b], Carlo Combi[b], Manfred Reichert[a]

[a] *Institute of Databases and Information Systems, Ulm University, Germany*
[b] *Department of Computer Science, University of Verona, Italy*

Abstract

The proper handling of temporal process constraints is crucial in many application domains. A sophisticated support of time-aware processes, however, is still missing in contemporary information systems. As a particular challenge, temporal constraints must be also handled for modularized processes (i.e., processes comprising subprocesses), enabling the reuse of process knowledge as well as the modular design of complex processes. This paper focuses on the representation and support of such time-aware modularized processes. In particular, we present a sound and complete method to derive the duration restrictions of a time-aware (sub-)process in such a way that its temporal properties are *completely* specified. We then show how this can be utilized when re-using the process within a modularized one. Altogether, the presented approach will foster the efficient and modular design of complex time-aware processes.

Keywords: Process-aware Information System, Temporal Constraints, Subprocess, Process Modularity, Controllability

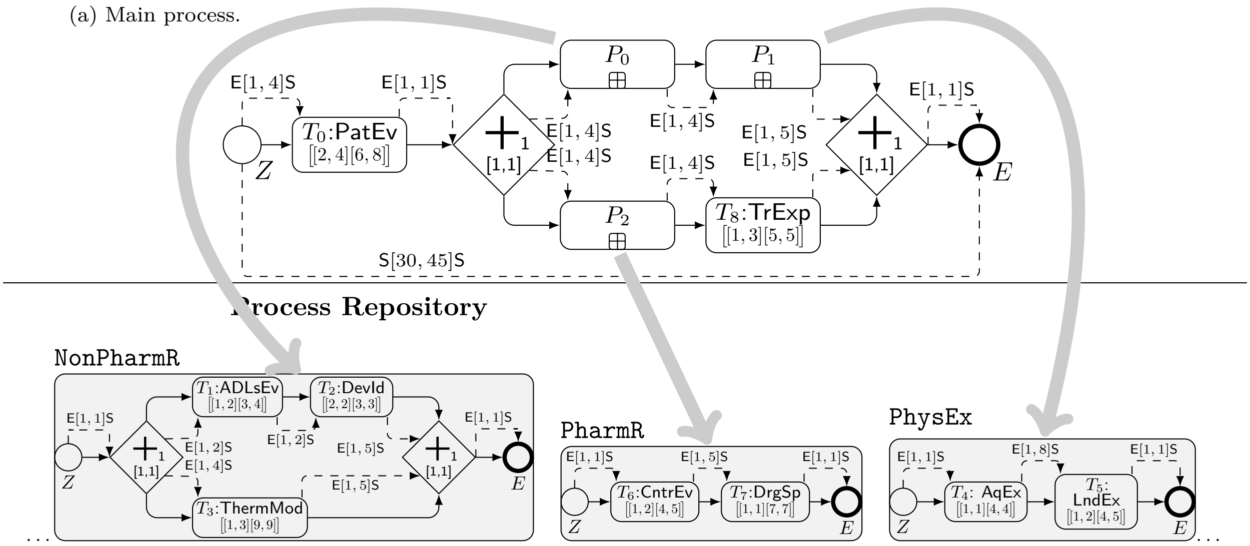
1. Introduction

The proper support of temporal process constraints is indispensable in many application domains. Although it has received increasing attention in the research community [4, 10, 12, 18], a sophisticated support of time-aware processes is still missing in contemporary process-aware information systems (PAIS). It is further widely acknowledged that the capability to modularly design process schemata constitutes a fundamental requirement for obtaining comprehensible and re-usable process schemas [23, 25]. Thus, the support of processes comprising subprocesses is essential as they allow for the reuse of existing process knowledge in a process repository as well as the modular design of complex processes.

At first glance, temporal process constraints and process modularity seem to be orthogonal features that may be managed in an independent way. When taking a closer view on them, however, it turns out that modularity in combination with the reuse of time-aware processes requires the ability to represent the overall temporal behavior of a process. This way, temporal constraints of a process containing time-aware subprocesses can be evaluated in a *true modular* way, i.e., without replacing the subprocess tasks with their (temporal) components. Moreover, it then becomes possible to attach such information to the process when storing it in a central process repository. This knowledge can, for example, be essential in the context of business process analysis and optimization [24].

To the best of our knowledge, the issue of representing the overall temporal properties of a process has not been considered in literature so far. This paper, therefore, focuses on the representation and support of time-aware modularized processes. In particular, we introduce a sound and complete method to derive the duration restrictions of a time-aware process in such a way that its temporal properties are *completely*

Email addresses: `andreas.lanz@uni-ulm.de` (Andreas Lanz), `roberto.posenato@univr.it` (Roberto Posenato), `carlo.combi@univr.it` (Carlo Combi), `manfred.reichert@uni-ulm.de` (Manfred Reichert)



(b) Subprocess P_0. (c) Subprocess P_2. (d) Subprocess P_1.

Figure 1: Motivating example: The process for managing osteoarthritis.

described. Then, we show how this characterization of a process can be merged with other temporal constraints when re-using it as a subprocess of a modularized process. In accordance with recent research contributions, we focus on the *dynamic controllability* (DC) of time-aware processes [4, 12]. In general, DC corresponds to the capability of a process engine to execute a process schema for *all* allowed durations of *all* tasks, while still satisfying *all* temporal constraints; i.e., DC ensures that it is possible to execute a process schema without any need to restrict the allowed durations of a task to be able to satisfy one of the other temporal constraints. In this context, task durations are called *contingent* as they are not under the control of the process engine.

As a motivating scenario, consider a high-level specification of an excerpt of a clinical guideline related to the management of osteoarthritis of the hand, hip and knee [11]. The process considers several aspects from pharmacological therapies to non-pharmacological ones to the provision of assistive devices. A possible schema for this process is depicted in Fig. 1. After completing the initial *Patient Evaluation* (task T_0: `PatEv`) two parallel branches become activated. The first one is composed of process *Non-Pharmacologic Recommendation* (P_0: `NonPharmR`) followed by process *Specification of Physical Exercises* (P_1: `PhysEx`). The second one consists of process *Pharmacologic Recommendation* (P_2: `PharmR`) followed by a *Treatment Explanation* to the patient (task T_8: `TrExp`). As depicted in Fig. 1, P_0, P_1, and P_2 constitute subprocesses from a process repository which, in turn, are composed of other tasks and are re-usable in other clinical processes (e.g., related to other pathologies). In detail, *Non-Pharmacologic Recommendation* P_0 consists of two parallel branches: The first one evaluates the patient's ability to perform activities of daily live (task T_1: `ADLsEv`) followed by the identification of needed assistive devices (task T_2: `DevId`). The second branch consists of giving instructions to the patient related to the use of thermal modalities (task T_3: `ThermMod`). In turn, the *Specification of Physical Exercises* (i.e., P_1) consists of the specification of aquatic exercises (task T_4: `AqEx`) followed by the specification of land exercises (task T_5: `LndEx`). Finally, *Pharmacologic Recommendation* (i.e., P_2) consists of the evaluation of contraindications (task T_6: `CntrEval`) followed by a drug specification (task T_7: `DrgSp`).

We enrich these process schemas with temporal constraints that need to be obeyed to guarantee the successful completion of each step of the therapy. They allow for the temporal characterization of tasks, edges and gateways, according to the concepts introduced in [13]. Note that the durations of tasks are not completely under the control of the process engines as these tasks are carried out by human users (e.g., doctors, nurses). Therefore, task durations are represented as *guarded ranges*. Such a duration range may be partially restricted by the system during process execution to ensure successful completion of the processes.

For example, task T_6 has temporal constraint $[\![1,2][4,5]\!]$ meaning that prior to the execution of the task its duration may be restricted, but in any case the minimum required duration must not exceed 2 time units and the maximum duration cannot be constrained below 4 (e.g., a duration of $[3,5]$ or $[1,2]$ would not be allowed). As another example consider task T_7 with temporal constraint $[\![1,1][7,7]\!]$. The latter means that this task may last 1 to 7 time units and all possible durations shall be allowed during process execution. This ensures that the user executing the task has enough flexibility to successfully complete the task. Constraints on gateways and edges are standard temporal constraints, specifying the possible durations (within a range), which are under the control of the process engine. The two main research questions addressed in this paper are:

1. How can the overall temporal behavior of a process be represented (cf. Sect. 3)? Addressing this question is a fundamental prerequisite for being able to provide some kind of modularity from the temporal perspective as well. *Note that without such characterization, it would be necessary to re-compute the temporal features of a subprocess each time it is used in a modularized process.* As will be shown, a subprocess can be represented as a kind of extended guarded range. On one hand the duration of the subprocess can be controlled to some extent due to the nature of the contained temporal constraints; on the other, it cannot be completely controlled since the contingent durations of the contained tasks must be guaranteed.

2. How to apply such knowledge when using a process as a subprocess inside a modularized process, in order to avoid having to re-analyze the internal constraints of the subprocess (Sect. 4)? This will, for example, enable us to store time-aware processes including their overall temporal properties inside a process repository and to reuse them in a truly modular fashion.

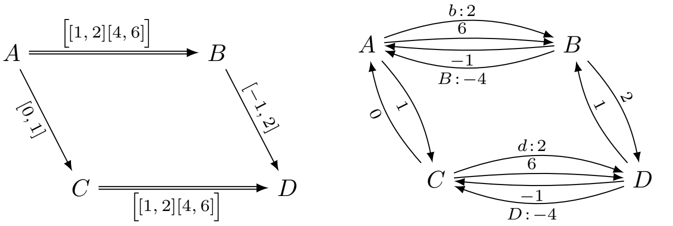
2. Background and Related Work

In literature, there exists considerable work on managing temporal constraints for business processes [1, 2, 5, 8, 12]. These approaches focus on issues like the modeling and verification of time-aware processes. Most of them use a specifically tailored time model to check for the temporal consistency of process schemas [19]. In [8, 9], an extended version of the *Critical Path Method* known from project planning is used. *Simple Temporal Networks with Uncertainty* (STNU) [21] are used as basic formalism in [5], whereas authors in [1, 3, 12] use *Conditional Simple Temporal Networks with Uncertainty* for checking the DC of process schemas. This paper relies on *Simple Temporal Network with Partially Shrinkable Uncertainty* (STNPSU), an extension of STNU where contingent links are extended for a more flexible management of temporal constraints [13].

An STNPSU [13] is a directed weighted graph (cf. Fig. 2a) where nodes represent time-point variables (timepoints), usually corresponding to the start or end of activities, and edges $A \xrightarrow{[x,y]} B$, called *requirement links*, represent a lower and an upper bound constraint on the distance between the two timepoints it connects; e.g., $A \xrightarrow{[x,y]} B$ represents the constraint that timepoint B has to occur between x and y time units after the occurrence of A (i.e., $x \leq B - A \leq y$). In an STNPSU, it is possible to characterize certain timepoints as *contingent timepoints*, meaning that their value cannot be decided by the system executing the STNPSU, but is decided by the environment at run time. Each contingent timepoint has one incoming edge, called *guarded link*, drawn with a double line, e.g., $A \xRightarrow{[\![x,x'][y',y]\!]} C$. A guarded link $A \xRightarrow{[\![x,x'][y',y]\!]} C$ consists of a pseudo-contingent duration range $[x,y]$ augmented with two *guards*, the *lower guard* x' and the *upper guard* y' [13]. A is called the *activation timepoint.* Before executing a guarded link, its duration range $[x,y]$ can be modified. However, any modification must be done in a way respecting the corresponding guards, i.e., $x \leq x'$ and $y \geq y'$. When activating a guarded link $A \xRightarrow{[\![x,x'][y',y]\!]} C$ (i.e., when executing timepoint A), the current value $[x^*, y^*]$ of the duration range $[x,y]$ becomes a fully contingent range, which is then made available to the environment for executing timepoint C. That is, once A is executed, C is guaranteed to be executed such that $C - A \in [x^*, y^*]$ holds. However, the particular time at which C is executed is *uncontrollable* since it is decided by the environment; i.e., it can be only observed when it happens.

More formally, an STNPSU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{G})$, where

- $\mathcal{T}$ is a set of *timepoints*;



(a) Non-DC STNPSU

(b) Distance Graph of the STNPSU in (a)

Figure 2: Non-DC STNPSU and corresponding distance graph.

- $\mathcal{C}$ is a set of *requirement links* $X \xrightarrow{[u,v]} Y$; and
- $\mathcal{G}$ is a set of *guarded links* each having the form $A \xRightarrow{[[x,x'][y',y]]} C$ where A and C are timepoints, and $0 < x \leq y < \infty$, $x \leq x'$, $0 < y' \leq y$.

Moreover, if $A_1 \xRightarrow{[[x_1,x_1'][y_1',y_1]]} C_1$ as well as $A_2 \xRightarrow{[[x_2,x_2'][y_2',y_2]]} C_2$ are distinct guarded links in $\mathcal{G}$, then C_1 and C_2 are distinct timepoints. It is noteworthy that guarded links may be used to represent two different types of constraints: If $x' < y'$ holds, a guarded link represents a temporal constraint with a *partially contingent range*. Particularly, the guarded link represents a constraint with a *contingent* (i.e., *unshrinkable*) *core* $[x', y'] \subseteq [x, y]$. In turn, if $x' \geq y'$ holds, a guarded link represents a temporal constraint with a *partially shrinkable range* with a *guarded core* $[y', x']$. In detail, this represents a constraint whose bounds cannot be shrunk beyond a certain point (i.e., x' and y', respectively). As opposed to a contingent link, x may be restricted to be greater than y' and y to be lower than x'.

Furthermore, each STNPSU is associated with a *distance graph* $\mathcal{D} = (\mathcal{T}, \mathcal{E})$, derived from the upper and lower bound constraints [13, 21]. In the distance graph(cf. Fig. 2b), each link between a pair of timepoints A and B is represented as two *ordinary edges* in $\mathcal{E}$: $A \xrightarrow{y} B$, representing the constraint $B \leq A + y$, and $A \xleftarrow{-x} B$, for the constraint $B \geq A + x$, $x, y \in \mathbb{R}$. Moreover, for each guarded link between a pair of timepoints A and C, $\mathcal{E}$ contains two other labeled edges, called *lower* and *upper case labeled values*. A lower case labeled value, $A \xrightarrow{c:x'} C$, represents the fact that C cannot be forced to be executed at a time greater than x' after A, i.e., it is not possible to add a constraint $A \xleftarrow{-x''} C, x' < x''$ to the network. In turn, an upper case labeled value, $A \xleftarrow{C:-y'} C$, represents the fact that C cannot be forced to be executed at a time less than y' after A, i.e., it is not possible to add a constraint $A \xrightarrow{y''} C, y'' < y'$ to the network.

These two kinds of labels are fundamental for determining the dynamic controllability of the network as explained in the following. Note that these two representations of an STNPSU can be used interchangeably.

An STNPSU is denoted as *dynamically controllable* (DC), if there exists a strategy for executing its timepoints in such a way that: i) all constraints in the network can be satisfied, no matter how the execution of any guarded link turns out, and ii) for any other guarded link $A \xRightarrow{[[x,x'][y',y]]} C$ the lower bound x never must be increased beyond its lower guard x' and the upper bound y never must be decreased below its upper guard y' [13].

In [13], the authors showed how one can adapt and extend the edge-generation rules and algorithm proposed by Morris et al. for checking the DC of STNU [21], called MM5, in order to check the dynamic controllability of an STNPSU in polynomial time (cf. Alg. 1). The checking algorithm works by recursively generating new edges in the STNPSU distance graph according to the rules from Table 1 and checking whether newly added edges determine so called *negative semi-reducible cycles* in the graph [20]. For each rule, existing edges are represented as solid arrows and newly ones as dashed arrows. Each of the first four rules takes two existing edges as input and generates a single edge as output. Finally, notation $R \not\equiv S$ expresses that R and S must be distinct time-point variables, and does not represent a constraint on the *values* of those variables. A path in an STNPSU distance graph is called semi-reducible if, by subsequent application of the edge generation rules (cf. Table 1), it can be transformed into a path solely consisting of ordinary or upper-case edges [20]. A semi-reducible cycle with negative unlabeled length is called a negative

Table 1: Edge-generation rules of the STNPSU-DC-Check algorithm. Dashed edges are the generated ones.

Rule	Edges	Applicable if
No Case:	$Q \xrightarrow{u} S \xrightarrow{v} T$, $\; Q \overset{u+v}{\dashrightarrow} T$	
Upper Case:	$Q \xrightarrow{u} S \xrightarrow{R\,:\,v} T$, $\; Q \overset{R\,:\,u+v}{\dashrightarrow} T$	
Lower Case:	$Q \xrightarrow{s\,:\,u} S \xrightarrow{v} T$, $\; Q \overset{u+v}{\dashrightarrow} T$	$v < 0 \vee (v = 0 \wedge S \not\equiv T)$
Cross Case:	$Q \xrightarrow{s\,:\,u} S \xrightarrow{R\,:\,v} T$, $\; Q \overset{R\,:\,u+v}{\dashrightarrow} T$	$R \not\equiv S \wedge (v < 0 \vee (v = 0 \wedge S \not\equiv T))$
Label Removal:	$S \xrightarrow{R\,:\,v} T$, $\; S \overset{v}{\dashrightarrow} T$	$R \not\equiv S \wedge v \geq -x$, $-x$ is the lower bound of the guarded link from T to R

Algorithm 1: STNPSU-DC-Check(G)

```
Input: G = (𝒯, 𝒞, 𝒢): STNPSU graph instance to analyze.
Output: the dynamic controllability of G.
D := distance graph of G;
for 1 to CutOffBound do // CutOffBound=O(|𝒯|)
    D' := AllMax-Projection of D;
    if (D' has a negative cycle) then return false;
    Generate new edges in D using edge-generation rules from Table 1;
    if (no edges generated) then return true;
return false;
```

semi-reducible cycles. To detect negative semi-reducible cycles Alg. 1 uses the *AllMax-Projection* of the STNPSU. That is, it gathers the ordinary and upper-case edges–without their labels–into a *simple temporal network* (STN) [7] and then checks this STN for consistency (e.g., using an all-pairs-shortest-path algorithm).

Example 1 (Negative Semi-Reducible Cycle)**.** As example, consider the distance graph depicted in Fig. 2b corresponding to the STNPSU in Fig. 2a. It is a matter of applying the edge generation rules from Table 1 to verify that the following corresponds to a semi-reducible cycle of the network (dashed lines are the generated ones):

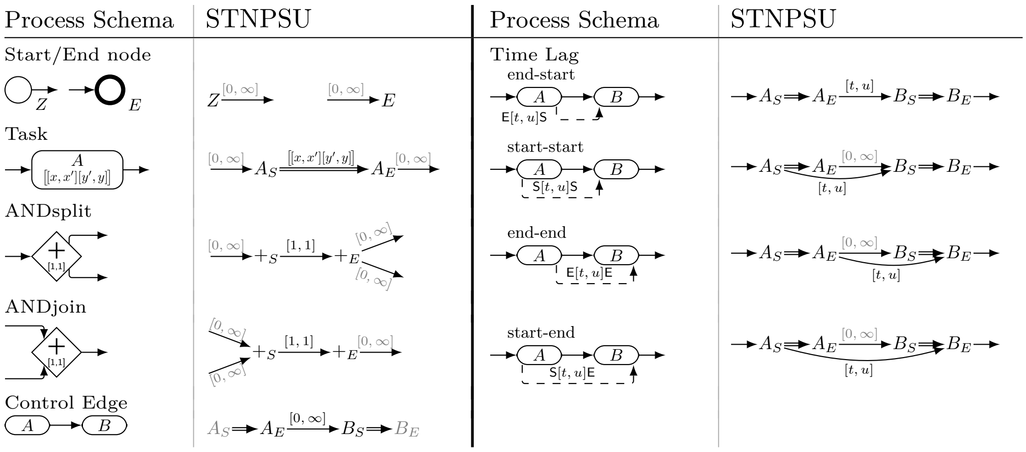
$$\begin{tikzcd} A \arrow[r, "b:2"] \arrow[rrr, dashed, bend right, "0"'] & B \arrow[r, "2"] \arrow[rr, dashed, bend left, "D:-2"] & D \arrow[r, "D:-4"] & C \arrow[r, "d:2"] \arrow[rrr, dashed, bend right, "B:-1"'] & D \arrow[r, "1"] \arrow[rr, dashed, bend left, "B:-3"] & B \arrow[r, "B:-4"] & A \end{tikzcd}$$

Moreover, as the unlabeled length of this semi-reducible cycle is negative the respective STNPSU cannot be DC. In particular, in the scenario where D is executed 4 after C and B is executed 2 after A, C has to be executed at most 0 after (i.e., at the same time as) A to be able to satisfy the requirement link between B and D. In turn, in the scenario where D is executed 2 after C and B is executed 4 after A, C has to be executed at least 1 after A to be able to statisfy the requirement link between B and D. However, it is not possible to satisfy both conditions at the same time. Thus the STNPSU is not DC.

We observe that the edge-generation rules from Table 1 only generate ordinary or upper-case edges. The upper-case edges generated by respective rules represent conditional constraints, called *waits* [22]. In particular, an upper-case edge $B \xrightarrow{C:-v} A$ represents the following constraint: as long as contingent timepoint C remains unexecuted, timepoint B must wait at least v units after the execution of A, the activation timepoint for C.

For each process exhibiting temporal constraints, a *time-aware process schema* needs to be defined [12]. In the context of this work, a process schema corresponds to a directed graph that comprises a set of *nodes*—representing *tasks* and *gateways* (e.g., AND-Split/Join)—as well as a set of *control edges* linking these nodes and specifying precedence relations between them. Each process schema contains a unique start and end node, and may be composed of control flow patterns like sequence, parallel split, and synchronization.

Table 2: STNPSU transformation rules.



In [18, 17], we introduce 10 time patterns representing common temporal constraints of time-aware processes. In particular, time patterns facilitate the comparison of existing approaches based on a universal set of notions with well-defined semantics [16]. Moreover, [18] elaborated the need for a proper run-time support of time-aware processes. In this work, we focus on the most fundamental category of time patterns, i.e., *durations* and *time lags*.

3. Characterization of Time-Aware Processes

This section shows how to determine a proper representation for the duration of a process. For this purpose, we consider a process schema P with a single start and a single end node. Note that in this paper we do not consider the choices pattern, but we are currently extending STNPSU to support choices as well. Moreover, preliminary analysis shows that the results presented in this paper will be applicable to this extended kind of STNPSU. First, we show how to verify the *dynamic controllability* (DC) of process schema P and, if P is DC, how to derive its minimal constraints. Next, we show how to determine the guards for a guarded link representing the duration of a process. Finally, we propose to extend the concept of guarded range in order to completely represent the overall temporal properties of a process.

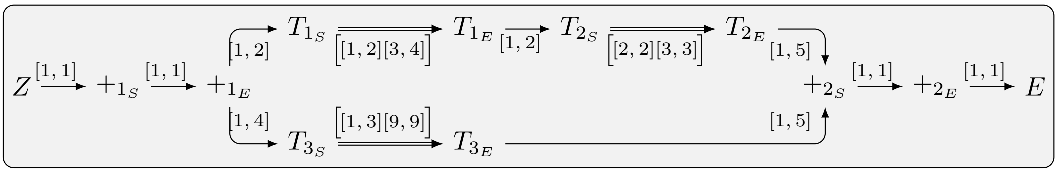
3.1. STNPSU Representation of a Process Schema

In order to verify the dynamic controllability of a process schema P, it is transformed into an STNPSU S using the transformation rules depicted in Table 2. The resulting STNPSU is characterized by having a single *initial timepoint* that occurs before any other one—called Z—and a single *ending timepoint*—called E—that occurs after any other timepoint. This STNPSU is then checked for DC by applying the standard algorithm for DC checking [13]. Given above transformation, it is easily possible to show that the process results to be DC if and only if the corresponding STNPSU is DC, which gives rise to the following theorem.

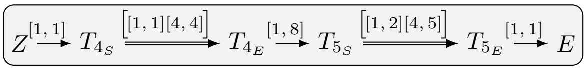
Theorem 1. *Given a time-aware process schema P built considering the process modeling elements depicted in Table 2, there exists a STNPSU S_P such that P is dynamically controllable if and only if S_P is DC.*

Proof. Table 2 depicts the mapping of the considered process modeling elements that can be used to build a time-aware process—tasks, control edges, AND gateways and temporal constraints—to the associated STNPSU fragments.

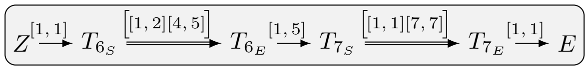
– `Activity`. Given a process schema, each task node A is transformed into two STNPSU timepoints, A_S and A_E, representing its start and end instants. The duration attribute of A, $[[x, x'][y', y]]$, is converted to the guarded link $A_S \xRightarrow{[[x,x'][y',y]]} A_E$.



(a) STNPSU corresponding to P_0.



(b) STNPSU corresponding to P_1.



(c) STNPSU corresponding to P_2.

Figure 3: STNPSUs corresponding to subprocesses P_0, P_1 and P_2 depicted in Fig. 1.

– `ANDjoin/ANDsplit` gateways. The conversion process is analogous to the one of a task. In this case, however, duration attribute $[x, y]$ is converted to a requirement link $A_S \xrightarrow{[x,y]} A_E$, as control connectors are executed by the process engine.

– `Control Edge`. A control edge from task A to task B is converted to a requirement link $A_E \xrightarrow{[0,\infty]} B_S$ with duration range $[0, \infty]$ in order to guarantee the right execution order of the original process.

– `Time Lags`. Consider a time lag $\langle \mathsf{I_F} \rangle [\mathsf{t}, \mathsf{u}] \langle \mathsf{I_S} \rangle$, where I_F and I_S represent the kind of instants to be considered, i.e., 'S' for the start instant, and 'E' for the end one. If the considered time lag is between tasks A and B, it is converted to a requirement link between the timepoints associated to instants A_{I_F} and B_{I_S} of the two tasks A and B. The resulting requirement link has the same duration range $[t, u]$ as the time lag.

Let P be a time-aware process schema. Applying the above transformation to P and to the possible time lags, one can simply verify that the obtained STNPSU represents all precedence relations and temporal constraints of original process schema P.

As introduced in Sect. 1, a time-aware process schema is *dynamically controllable* if it is possible to execute it for all required durations of all activities, while still satisfying all temporal constraints. Furthermore, we recalled that an STNPSU is *dynamically controllable* if it is possible to execute it in a way such that, no matter how the execution of any guarded link turns out, for any other guarded link $A \xrightarrow{[[x,x'][y',y]]} C$ the lower bound x never must be increased beyond its guard x' and the upper bound y never must be decreased below its guard y' in order to ensure controllability of the network.

Therefore, it is a matter of definitions to verify that the dynamic controllability of a process schema implies the dynamic controllability in the corresponding STNPSU and vice versa. □

3.2. Lower and Upper Guard

Note that the DC checking algorithm also derives the minimum and maximum duration between timepoints Z and E, i.e., the minimum and maximum durations of the process. However, these bounds are not sufficient for characterizing the temporal behavior of the process as they do not represent its possible non-restrictable duration ranges. As an example consider the STNPSU depicted in Fig. 3c, which corresponds to process P_2 of Fig. 1. One can easily show that the duration range between Z and E corresponds to $[5, 19]$. However, this range cannot be reduced to $[5, 10]$, for example, since the internal task T_7 has a contingent duration of 1 to 7, which cannot be controlled (i.e., restricted) by the process engine. In particular, if T_7 lasts exactly 7, process P_2 lasts at least 11 time units. On the other hand, representing a subprocess by considering the duration range between Z and E to be a contingent one would make the overall process over-constrained, and thus limit the overall temporal flexibility of the modularized process.

We, therefore, suggest representing the duration of a process by a guarded range with proper guards in order to prevent unacceptable restrictions of the duration range of the process. In the following, we propose a method to determine the lower and upper guard of such guarded range based on the STNPSU representation of the process schema. In this context, the *upper guard* for the duration range of a process P represents the lowest value the maximum duration of the process may be decreased to. In other words, considering the corresponding STNPSU S of P, the upper guard corresponds to the lowest value the upper bound of the requirement link, which is derived between Z and E by the DC checking algorithm, may be decreased to. It can be determined considering the maximum guards of any guarded link and the lower bounds of any requirement link in S as outlined in Example 2.

Example 2 (Upper Guard)**.** Consider the STNPSU depicted in Fig. 3c. While the upper bounds of the internal requirement links may be restricted to their lower bounds (i.e., 1) by the process engine, the upper bounds of the two guarded links cannot be restricted below their upper guards (i.e., 4 and 7, respectively). Therefore, the value we obtain when summing the lower bound values of the requirement links and the upper guards of the guarded links, i.e., $1+4+1+7+1=14$, represents the minimal value the upper bound of the link between Z and E may be restricted to.

In turn, the *lower guard* for the duration range of a process P represents the greatest value the minimum duration of the process may be increased to. In the STNPSU S, therefore, the lower guard corresponds to the greatest value the lower bound of the requirement link between Z and E may be increased to.

If there are several paths leading from Z to E, it is necessary to consider the maximum/minimum such value considering all paths. Therefore, Defs 1 and 2 specify the concept of lower/upper guard for any timepoint of an STNPSU.

Definition 1 (Upper Guard)**.** Given a dynamically controllable STNPSU S with distance graph $\mathcal{D}=(\mathcal{T},\mathcal{E})$ and a timepoint C. Then: The minimum value that may be set for the upper bound v of a requirement link $Z\xrightarrow{[u,v]}C$ is called the *upper guard* of C:

$$\text{upperGuard}_S(C)=\max_{B\in\mathcal{T}}\begin{cases}0 & \text{if } Z\equiv C\\ \text{upperGuard}_S(B)+x & \text{if } (B\xleftarrow{-x}C)\in\mathcal{E}\\ \text{upperGuard}_S(B)+y' & \text{if } (B\xleftarrow{D:-y'}C)\in\mathcal{E}\end{cases}$$

Definition 2 (Lower Guard)**.** Given a dynamically controllable STNPSU S with distance graph $\mathcal{D}=(\mathcal{T},\mathcal{E})$ and a timepoint C. Then: The maximum value that may be set for the lower bound u of a requirement link $Z\xrightarrow{[u,v]}C$ is called the *lower guard* of C:

$$\text{lowerGuard}_S(C)=\min_{B\in\mathcal{T}}\begin{cases}0 & \text{if } Z\equiv C\\ \text{lowerGuard}_S(B)+y & \text{if } (B\xrightarrow{y}C)\in\mathcal{E}\\ \text{lowerGuard}_S(B)+x' & \text{if } (B\xrightarrow{d:x'}C)\in\mathcal{E}\end{cases}$$

Considering Defs. 1 and 2 it is easy to verify that:

- the upperGuard of a requirement link $A\xrightarrow{[x,y]}C$ is x;
- the upperGuard of a guarded link $A\xrightarrow{[[x,x'][y',y]]}C$ is y';
- the lowerGuard of a requirement link $A\xrightarrow{[x,y]}C$ is x;
- the lowerGuard of a guarded link $A\xrightarrow{[[x,x'][y',y]]}C$ is y';
- in general, for any timepoints A and C with $Z\xrightarrow{[a,b]}A\xrightarrow{[c,d]}C$ derived by the DC checking algorithm, it holds $\text{upperGuard}(C)\geq\text{upperGuard}(A)+c$ and $\text{lowerGuard}(C)\leq\text{lowerGuard}(A)+d$.

Example 3. Regarding the STNPSUs depicted in Fig. 3, one can verify that the values of lowerGuard and upperGuard between Z and E correspond to

- $\text{lowerGuard}_{P_0}(E)=15$ and $\text{upperGuard}_{P_0}(E)=15$,
- $\text{lowerGuard}_{P_1}(E)=13$ and $\text{upperGuard}_{P_1}(E)=11$, and
- $\text{lowerGuard}_{P_2}(E)=10$ and $\text{upperGuard}_{P_2}(E)=14$.

Definitions 1 and 2 allow determining to which extent the upper/lower bound of the derived requirement link between Z and a timepoint C in an STNPSU S may be reduced/increased, without affecting the DC of S (cf. Lemmas 1 and 2).

Lemma 1 (Upper Guard)**.** *Let S be a dynamically controllable STNPSU, Z be the initial timepoint and C be a timepoint in S. Then: The upper bound v of the distance $Z \xrightarrow{[u,v]} C$ between Z and C may be reduced to at most* $\mathrm{upperGuard}_S(C)$*, preserving the DC of S.*

Proof. First, we show that if v is set to be less than $\mathrm{upperGuard}_S(C)$, then the network cannot be DC. Let $B_1 \dots B_k$ be the path from Z to C in the distance graph $\mathcal{D}$ that determines the value for $\mathrm{upperGuard}(C)$, i.e.,

$$Z \xleftarrow{\alpha_0} B_1 \xleftarrow{\alpha_1} \dots \xleftarrow{\alpha_{k-1}} B_k \xleftarrow{\alpha_k} C$$

where α_i is either an ordinary or upper case edge and $-\sum_{i\in\{0,\dots,k\}} \tilde{\alpha_i} = \mathrm{upperGuard}(C)$, with $\tilde{\alpha_i}$ corresponding to the value of α_i ignoring any label. Given such path, in the AllMax-Projection $\mathcal{D}'$ any upper case edge $\alpha_i = \{D_i : -y_i'\}$ is replaced by $\tilde{\alpha_i} = -y_i'$. Thus it is easy to verify that by the standard STN propagation rules in the AllMax-Projection an ordinary edge $Z \xleftarrow{\sum\tilde{\alpha_i}} C$ is derived. At the same time if we add a requirement edge $Z \xrightarrow{v^*} C$ with $v^* < \mathrm{upperGuard}(C)$ to the distance graph $\mathcal{D}$ of the original STNPSU S the same edge will also be added to the AllMax-Projection $\mathcal{D}'$, determining a negative cycle $Z \xleftarrow{\sum\tilde{\alpha_i}} C \xleftarrow{v^*} Z$, i.e., the STNPSU cannot be DC.

Second, we show that if S is DC and v is reduced to a value $v' \geq \mathrm{upperGuard}_S(C)$ then v' cannot be part of any negative semi-reducible cycle, i.e., the resulting network must be DC as well. Let us assume that $Z \xrightarrow{[u,v]} C$ is restricted to $Z \xrightarrow{[u,v']} C$ with $\mathrm{upperGuard}(C) \leq v' \leq v$ and that the resulting network is not DC. This implies that there exists a negative semi-reducible cycle $Z \xleftarrow{\alpha_0} E_1 \xleftarrow{\alpha_1} \dots \xleftarrow{\alpha_{l-1}} E_l \xleftarrow{\alpha_l} C \xleftarrow{v^*} Z$ in the distance graph $\mathcal{D}$ consisting only of ordinary or upper case edges α_i such that $\sum_{i\in\{0,\dots,l\}} \tilde{\alpha_i} + v^* < 0$, i.e., $v^* < -\sum_{i\in\{0,\dots,l\}} \tilde{\alpha_i}$. Based on Def. 1 it then follows that for any such path $E_1, \dots, E_l$ from Z to C it holds $\mathrm{upperGuard}(C) \geq -\sum_{i\in\{0,\dots,l\}} \tilde{\alpha_i}$ and thus $\mathrm{upperGuard}(C) \leq v^* < -\sum_{i\in\{0,\dots,l\}} \tilde{\alpha_i} \leq \mathrm{upperGuard}(C)$ which contradicts the assumption. □

Lemma 2 (Lower Guard)**.** *Let S be a dynamically controllable STNPSU, Z be the initial timepoint and C be a timepoint in S. Then: The lower bound u of distance $Z \xrightarrow{[u,v]} C$ between Z and C may be increased to at most* $\mathrm{lowerGuard}_S(S)$*, preserving the DC of S.*

Proof. The proof is analogous to the proof of Lemma 1 using the AllMin-Projection and similar reasoning. The AllMin-Projection is similar to the AllMax-Projection, but considering only ordinary and lower-case edges. □

Using Defs 1 and 2, it now becomes possible to determine to which extent the lower/upper bound of the duration range of a process can be restricted, while preserving its DC as illustrated by Example 4.

Example 4. The minimum and maximum durations of the processes depicted in Fig. 1 are determined by the DC checking algorithm as P_0: $[11, 20]$, P_1: $[5, 19]$, and P_2: $[5, 19]$. Using Defs 1 and 2, it now becomes possible to determine to which extent these duration ranges may be restricted:

- the minimum duration of P_0 may be restricted to $\mathrm{lowerGuard}_{P_0}(E) = 15$ at most, while its maximum duration may be restricted to $\mathrm{upperGuard}_{P_0}(E) = 15$;
- the duration of P_1 may be restricted to $\mathrm{lowerGuard}_{P_1}(E) = 13$ and $\mathrm{upperGuard}_{P_1}(E) = 11$, respectively; and
- the duration of P_2 to $\mathrm{lowerGuard}_{P_2}(E) = 10$ and $\mathrm{upperGuard}_{P_2}(E) = 14$.

Based on the definitions of lowerGuard and upperGuard, one can easily verify that their value is always non-negative. Moreover, it is easy to verify that the $\mathrm{upperGuard}(C)$ value is given by value u of edge $Z \xleftarrow{-u} C$ in the AllMax-Projection graph of the network, while $\mathrm{lowerGuard}(C)$ value is given by value v of edge $Z \xrightarrow{v} C$

in the AllMin-Projection graph. Using standard STN algorithms [7], therefore, the computational cost of determining lowerGuard(C) and upperGuard(C) is at most $O(n^3)$, with n being the number of timepoints in the considered STNPSU.

3.3. Contingency Span

Given a range $[u, v]$ that represents the overall duration of a DC process, Defs. 1 and 2 assure that it is always possible to reduce one of the two bounds of the respective duration range to the corresponding guard (i.e., upperGuard(E) or lowerGuard(E)) without affecting the DC of the process. However, it is not possible to restrict both bounds simultaneously since the restriction of one bound may change the guard of the other bound as shown by Example 5.

Example 5. Let us consider the STNPSU from Fig. 3c that corresponds to subprocess P_2. One can easily determine that $\text{lowerGuard}_{P_2}(E) = 10$ and $\text{upperGuard}_{P_2}(E) = 14$ hold. Moreover, the duration range of the process is $[5, 19]$ as determined by the DC checking algorithm. Considering Lemmas 1 and 2, it then can be easily shown that the minimum duration of the process may be increased to 10 or its maximum duration may be restricted to 14. However, for process P_2 it is not possible to increase the minimum duration to 10, while at the same time restricting the maximum duration to 14. In particular, if the minimum duration is increased to 10, due to the *partially contingent guarded link* between timepoints T_{7_S} and T_{7_E} (representing task T_7), the maximum duration must not be decreased below 16 to further guarantee the DC of the process. On the other hand, the maximum duration may be decreased to 14, but then the minimum duration must not be increased beyond 8. In detail, a span of at least 6 must be ensured for the final duration range of the process.

To fully represent the overall temporal properties of a process we suggest considering an additional value that represents the minimal span to be guaranteed for the duration range. We denote this value as the *contingency span* of the process. It can be defined using the *link contingency span* and *path contingency span* of the corresponding STNPSU.

Definition 3 (Link Contingency Span)**.** A positive link contingency span Δ corresponds to the span that needs to be guaranteed for a link in order to ensure the DC of an STNPSU. In turn, a negative link contingency span corresponds to the maximum span provided by a link that can be used to reduce the contingency span of previous guarded link.

a) For a guarded link $A \xrightarrow{[[a,a'][b',b]]} B$, the link contingency span Δ_{AB} is defined as $\Delta_{AB} = b' - a'$.
b) For a requirement link $A \xrightarrow{[a,b]} B$, the link contingency span Δ_{AB} is defined as $\Delta_{AB} = a - b$.

Considering Def. 3 it is easy to verify that:

- the link contingency span of a requirement link is less than or equal to zero, i.e., $A \xrightarrow{[a,b]} B \Rightarrow \Delta_{AB} \leq 0$;
- the link contingency span of a *partially shrinkable guarded link* is less than or equal to zero, i.e., $A \xrightarrow{[[a,a'][b',b]]} B \wedge a' \geq b' \Rightarrow \Delta_{AB} \leq 0$;
- the link contingency span of a *partially contingent guarded link* is greater than zero, i.e., $A \xrightarrow{[[a,a'][b',b]]} B \wedge a' < b' \Rightarrow \Delta_{AB} > 0$.

Next, we need to find a way to determine the contingency span of a path based on the link contingency span of its links. First, let us consider a guarded link $A \xrightarrow{[[a,a'][b',b]]} B$ followed by a requirement link $B \xrightarrow{[c,d]} C$. In this case, the contingency span required by the guarded link can be partially or fully compensated by the subsequent requirement link, as the duration of the latter can be decided based on the actual duration of the former. Thus, the contingency of the path from A to C is given by $\Delta_{AB} + \Delta_{BC}$. In turn, for a requirement link $A \xrightarrow{[a,b]} B$ followed by a guarded link $B \xrightarrow{[[c,c'][d',d]]} C$ we must differentiate two subcases: If the guarded link is *partially contingent* (i.e., $c' < d'$) the previous requirement link cannot be used to compensate its contingency span as the duration of the requirement link must be decided before executing the guarded link. Therefore, the contingency span of the path from A to C is given by Δ_{BC}. However, if the guarded link is *partially shrinkable* (i.e., $d' \leq c'$), its link contingency Δ_{BC} is negative. In this case, the contingency span of the path from A to C is again given by $\Delta_{AB} + \Delta_{BC}$ as both links could be used to reduce the contingency of a previous guarded link. Finally, the combination of two requirement links (guarded links) is similar to the above cases. When considering a path that consists of more than two links, the link contingency spans need to be combined in an incremental way starting from the inital timepoint Z. When considering two or more parallel paths, in turn, it becomes necessary to consider the most demanding case, i.e., the path with the largest contingency span. This leads to the following recursive approach for calculating the contingency span of a path.

Definition 4 (Path Contingency Span)**.** Let S be a dynamically controllable STNPSU and Z be its initial timepoint. By definition the *path contingency span* of Z is $\mathrm{cont}_S(Z) = 0$. Then: The *path contingency span* $\mathit{cont}_S(C)$ of any other timepoint C is given by

$$\mathrm{cont}_S(C) = \max\left\{0, \max_{B \in \mathcal{T}} \{\mathrm{cont}_S(B) + \Delta_{BC}\}\right\}$$

It is noteworthy that the path contingency span of any timepoint is always greater or equal to zero, i.e., $\mathrm{cont}_S(C) \geq 0$. Moreover, the problem of determining the value of $\mathrm{cont}_S(C)$, i.e., the maximum contingency span among all possible paths from Z to C, can be reduced to the problem of finding the minimal distance between Z and C in a suitable weighted graph built considering the link contingency spans as edge values.

Definition 5 (Contingency Graph)**.** Let $S = (\mathcal{T}, \mathcal{C}, \mathcal{G})$ be an STNPSU to which the DC-checking algorithm has been applied (cf. Alg. 1). The corresponding *contingency graph* for S has the form $\mathcal{CO} = (\mathcal{T}, \mathcal{E}_{\mathcal{CO}})$. Thereby, each timepoint in $\mathcal{T}$ serves as a node in the graph; $\mathcal{E}_{\mathcal{CO}}$ is a set of weighted edges:

a) for each guarded link $A \xrightarrow{[[x,x'][y',y]]} B \in \mathcal{G}$ there exists a single edge $A \xrightarrow{-\Delta_{AB}} B \in \mathcal{E}_{\mathcal{CO}}$.
b) for each requirement link $A \xrightarrow{[x,y]} B \in \mathcal{C}$ there exist two edges $A \xrightarrow{-\Delta_{AB}} B, B \xrightarrow{-\Delta_{AB}} A \in \mathcal{E}_{\mathcal{CO}}$.
c) for each timepoint $T \in \mathcal{T}$ there exists an edge $Z \xrightarrow{0} T \in \mathcal{E}_{\mathcal{CO}}$.

Based on Def. 4 it is now easy to verify that the *path contingency span* of any timepoint $C \in \mathcal{T}$ corresponds to the negative value of the shortest path from initial timepoint Z to C in the corresponding *contingency graph* (cf. Def. 5).

It is worthy to note two things about Def. 4 and Def. 5. First, since a requirement link can connect two non sequential timepoints, its link contingency span can be used in combination with the contingency coming from any of its endpoints. Def. 5 considers these two mutually-exclusive possibilities by adding two edges $A \xrightarrow{-\Delta_{AB}} B, B \xrightarrow{-\Delta_{AB}} A \in \mathcal{E}_{\mathcal{CO}}$. Second, edges $Z \xrightarrow{0} T \in \mathcal{E}_{\mathcal{CO}}, T \in \mathcal{T}$ added by step c) in Def. 5 guarantee that the length of any path in the graph starting at timepoint Z is always less or equal to 0, i.e., the corresponding path contingency is always positive as requested by the definition.

Moreover, as S is DC, the contingency graph $\mathcal{CO}$ cannot contain any negative cycles. In particular, the only edges with negative edge value are the ones resulting from a partially contingent guarded link $A \xrightarrow{[[x,x'][y',y]]} B$. Then, for any path $B = E_0, \ldots, E_k = A$ it must hold $-\sum_{i=1\ldots k-1} \Delta_{E_i E_{i+1}} \geq \Delta_{AB}$, otherwise S cannot be DC. Using the Bellman–Ford algorithm [6], the computational cost of determining $\mathrm{cont}_S(C)$ is at most $O(n^3)$, with n being the number of timepoints in the STNPSU.

Example 6. The path contingency graph corresponding to the STNPSU depicted in Fig. 3a is shown in Fig. 4. Note that insignificant edges determined by the DC checking algorithm have been omitted for sake of readability. Applying the Bellman-Ford algorithm to this graph, the grayed values in bracket are determined (insignificant edges are again omitted). In particular, the edge ZE is derived as $Z \xrightarrow{-2} E$. Moreover, by applying Def. 4 to Fig. 3a it can be easily verified that $\mathrm{cont}_{P_0}(E) = 2$ holds.

Regarding the other STNPSUs from Fig. 3, the path contingency span of timepoints E are as follows:

- $\mathrm{cont}_{P_1}(E) = 2$, and
- $\mathrm{cont}_{P_2}(E) = 6$.

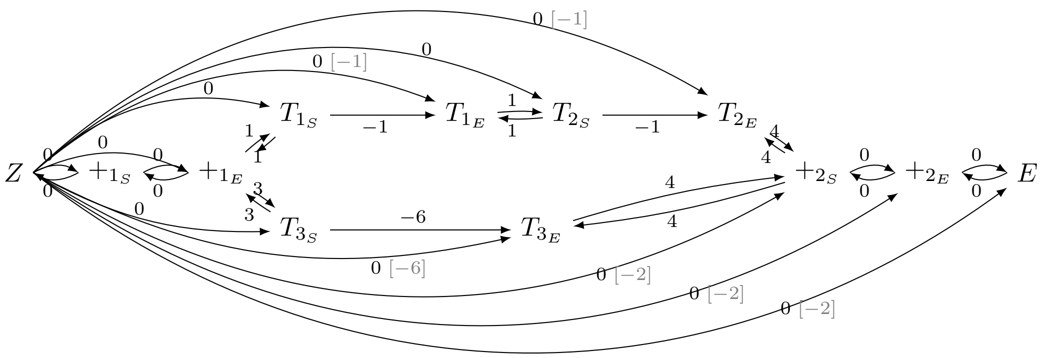


Figure 4: Contingency Graph of the STNPSU in Fig. 3a showing values determined by the Bellman-Ford algorithm (grayed bracketed values).

Based on Def. 4, it becomes possible to describe the admissible duration ranges between two timepoints in an STNPSU.

Lemma 3. *Let S be a dynamically controllable STNPSU, Z be its initial timepoint, and C be any other timepoint. Then: In order to preserve the DC of S, any restriction $Z \xrightarrow{[u^*, v^*]} C$ ($u \le u^* \le \text{lowerGuard}_S(C)$, $\text{upperGuard}_S(C) \le v^* \le v$) of the distance between Z and C must be done in such a way that $v^* - u^* \ge \text{cont}_S(C)$ holds.*

Proof. We are only interested in considering timepoints C with a positive path contingency span $\text{cont}_S(C) > 0$ and $\text{upperGuard}_S(C) - \text{lowerGuard}_S(C) < \text{cont}_S(C)$; otherwise it is already ensured that $v^* - u^* \ge \text{cont}_S(C)$ holds (either by the fact that $v^* - u^* \ge 0$ or by the guards).

First of all, let us consider the definition of $\text{cont}_S(C)$. Note that a positive path contingency span can only occur when there is at least one partially contingent guarded link inside S. Moreover, from the definition of $\text{cont}_S()$, it is always possible to find a sequence of timepoints $B_0, \ldots, B_k$ with $B_k \equiv C$ for which it holds

$$\text{cont}_S(C) = \text{cont}_S(B_0) + \Delta_{B_0,B_1} + \ldots + \Delta_{B_{k-1},B_k}$$

with

1. $\text{cont}_S(B_0) = 0$,
2. $\forall j \in \{1, \ldots, k\} : \sum_{i \in \{1,\ldots,j\}} \Delta_{B_{i-1},B_i} > 0$, i.e., $\forall j \in \{1, \ldots, k\} : \text{cont}_S(B_j) > 0$

Then, by definition, link B_0B_1 is a partially contingent guarded link: $B_0 \xRightarrow{\llbracket x_1, x'_1 \rrbracket \llbracket y'_1, y_1 \rrbracket} B_1$.

If path $B_0, \ldots, B_k$ contains a sequence of requirement links $B_{i-1} \xrightarrow{[x_i, y_i]} B_i \xrightarrow{[x_{i+1}, y_{i+1}]} B_{i+1}$ there also exists an equivalent single requirement links $B_{i-1} \xrightarrow{[x_i + x_{i+1}, y_i + y_{i+1}]} B_{i+1}$ resulting in the same value of $\text{cont}_S(B_{i+1})$. Moreover, if path $B_0, \ldots, B_k$ contains a sequence of guarded links $B_{i-1} \xRightarrow{\llbracket x_i, x'_i \rrbracket \llbracket y'_i, y_i \rrbracket} B_i \xRightarrow{\llbracket x_{i+1}, x'_{i+1} \rrbracket \llbracket y'_{i+1}, y_{i+1} \rrbracket} B_{i+1}$, it is always possible to split timepoint B_i into two timepoints B'_i and B''_i connected by a requirement link with value $[0, 0]$ without changing the properties of the network (i.e., in particular, $\text{cont}_S(B_{i+1})$), i.e., $B_{i-1} \xRightarrow{\llbracket x_i, x'_i \rrbracket \llbracket y'_i, y_i \rrbracket} B_i \xRightarrow{\llbracket x_{i+1}, x'_{i+1} \rrbracket \llbracket y'_{i+1}, y_{i+1} \rrbracket} B_{i+1} \equiv B_{i-1} \xRightarrow{\llbracket x_i, x'_i \rrbracket \llbracket y'_i, y_i \rrbracket} B'_i \xrightarrow{[0,0]} B''_i \xRightarrow{\llbracket x_{i+1}, x'_{i+1} \rrbracket \llbracket y'_{i+1}, y_{i+1} \rrbracket} B_{i+1}$.

In summary, without loss of generality we can assume that the sequence of timepoints $B_0, \ldots, B_k$ always has the following pattern:

$$Z \overset{[a,b]}{\rightsquigarrow} B_0 \xRightarrow{\llbracket x_1, x'_1 \rrbracket \llbracket y'_1, y_1 \rrbracket} B_1 \xrightarrow{[x_2, y_2]} B_2 \xRightarrow{\llbracket x_3, x'_3 \rrbracket \llbracket y'_3, y_3 \rrbracket} B_3 \xrightarrow{[x_4, y_4]} B_4 \ldots \xRightarrow{\llbracket x_{k-1}, x'_{k-1} \rrbracket \llbracket y'_{k-1}, y_{k-1} \rrbracket} B_{k-1} \xrightarrow{[x_k, y_k]} B_k \equiv C$$

where $Z \overset{[a,b]}{\rightsquigarrow} B_0$ is the requirement link derived by the DC checking algorithm.

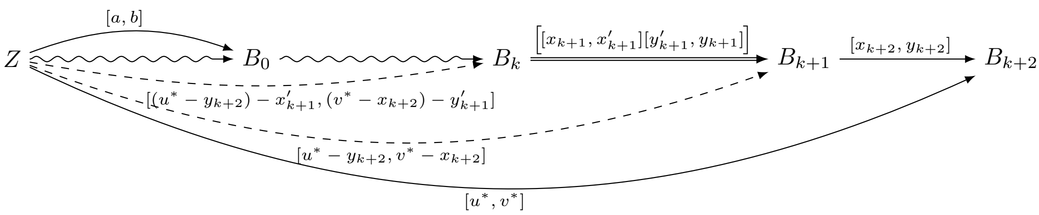
We can now show by induction that it is not possible to restrict $Z \xrightarrow{[u,v]} B_k$ to $[u^*, v^*]$ such that $v^* - u^* < \text{cont}_S(B_k)$. Particularly, assuming that $v^* - u^* = \text{cont}_S(B_k) - \epsilon$, $\epsilon > 0$ we show that at least one link inside the path $Z, B_0, \ldots, B_k$ has to be restricted beyond its bounds/guards.

First, consider a path consisting of 3 timepoints B_0, B_1, B_2, i.e., $Z \overset{[a,b]}{\rightsquigarrow} B_0 \xRightarrow{[[x_1, x_1'][y_1', y_1]]} B_1 \xrightarrow{[x_2, y_2]} B_2$ (Note that the case of two timepoints follows by assuming $y_2 = x_2 = 0$ and the case of one timepoints is given by definition because then $b - a \leq \text{cont}_S(B_0) - \epsilon =< 0$ holds). In this case $\text{cont}_S(B_2)$ is given by $\text{cont}_S(B_2) = \text{cont}_S(B_0) + (y_1' - x_1') + (x_2 - y_2)$, $\text{cont}_S(B_0) = 0$. Assume $Z \xrightarrow{[u,v]} B_2$ is restricted to $Z \xrightarrow{[u^*,v^*]} B_2$ with $v^* - u^* = \text{cont}_S(B_2) - \epsilon = (y_1' - x_1') + (x_2 - y_2) - \epsilon$, $\epsilon > 0$. Then, by the No-Case Rule (cf. Table 1) a requirement link $Z \xrightarrow{[u^* - y_2, v^* - x_2]} B_1$ between Z and B_1 is derived. Moreover, the Lower Case Rule derives an ordinary edge $B_0 \xrightarrow{x_1' - (u^* - y_2)} Z$. In turn, the Upper Case Rule derives a wait $B_0 \xrightarrow{B_1 : (v^* - x_2) - y_1'} Z$. This wait is transformed into ordinary edge $B_0 \xrightarrow{(v^* - x_2) - y_1'} Z$ by the Label Removal Rule because $(v^* - x_2) - y_1' \geq -x_1'$ holds, as $v^* \geq y_1' + x_2 \geq y_1' - x_1' + x_2$ must hold for the original network to be DC. In summary a requirement link $Z \xrightarrow{[(u^* - y_2) - x_1', (v^* - x_2) - y_1']} B_0$ is derived. Hence, it must hold $b \leq (v^* - x_2) - y_1'$ and $a \geq (u^* - y_2) - x_1'$ and, therefore, it must also hold

$$
\begin{aligned}
b - a &\leq (v^* - x_2) - y_1' - ((u^* - y_2) - x_1') \\
&= v^* - u^* + y_2 - x_2 + x_1' - y_1' \\
&= (y_1' - x_1') + (x_2 - y_2) - \epsilon + y_2 - x_2 + x_1' - y_1' \qquad {\scriptstyle v^* - u^* = \text{cont}_S(B_2) - \epsilon} \\
&= -\epsilon < 0
\end{aligned}
$$

which shows that the network can no longer be DC as the requirement link ZB_0 is restricted too much.

Now let us consider a path consisting of $k+3$ timepoints $B_0, \ldots, B_{k+2}$ as depicted below (Again, the case of $k+2$ timepoints follows by assuming $y_{k+2} = x_{k+2} = 0$).



Let us assume that $Z \xrightarrow{[u,v]} B_{k+2}$ is restricted to $Z \xrightarrow{[u^*,v^*]} B_{k+2}$ with $v^* - u^* = \text{cont}_S(B_{k+2}) - \epsilon$, $\epsilon > 0$. Then by the No-Case Rule (cf. Table 1) a requirement link $Z \xrightarrow{[u^* - y_{k+2}, v^* - x_{k+2}]} B_{k+1}$ is derived. Moreover, the Lower Case Rule derives an ordinary edge $B_k \xrightarrow{x_{k+1}' - (u^* - y_{k+2})} Z$. In turn, the Upper Case Rule derives a wait $B_k \xrightarrow{B_{k+1} : (v^* - x_{k+2}) - y_{k+1}'} Z$. This wait is transformed into ordinary edge $B_k \xrightarrow{(v^* - x_{k+2}) - y_{k+1}'} Z$ by the Label Removal Rule because $(v^* - x_{k+2}) - y_{k+1}' \geq -x_{k+1}'$ holds, as $v^* \geq y_{k+1}' + x_{k+2} \geq y_{k+1}' - x_{k+1}' + x_{k+2}$ must hold for the original network to be DC. In summary a requirement link $Z \xrightarrow{[(u^* - y_{k+2}) - x_{k+1}', (v^* - x_{k+2}) - y_{k+1}']} B_k$ is derived.

Thus for the span of the requirement link $Z \xrightarrow{[p,q]} B_k$ between Z and B_k derived by the DC checking algorithm it holds

$$
\begin{aligned}
q - p &\leq (v^* - x_{k+2}) - y_{k+1}' - ((u^* - y_{k+2}) - x_{k+1}') \\
&= (v^* - u^*) - (y_{k+1}' - x_{k+1}') - (x_{k+2} - y_{k+2}) \\
&= \text{cont}_S(B_{k+2}) - \epsilon - \Delta_{B_k B_{k+1}} - \Delta_{B_{k+1} B_{k+2}} \qquad {\scriptstyle v^* - u^* = \text{cont}_S(B_{k+2}) - \epsilon,\ \text{Def. 3}} \\
&= \text{cont}_S(B_k) + \Delta_{B_k B_{k+1}} + \Delta_{B_{k+1} B_{k+2}} - \epsilon - \Delta_{B_k B_{k+1}} - \Delta_{B_{k+1} B_{k+2}} \qquad {\scriptstyle \text{Def. 4}} \\
&= \text{cont}_S(B_k) - \epsilon
\end{aligned}
$$

Hence, the range of the requirement link $Z \xrightarrow{[p,q]} B_k$ is restricted such that $q - p \leq \mathrm{cont}_S(B_k) - \epsilon < \mathrm{cont}_S(B_k)$ holds. By subsequent application of the same steps (i.e., by induction) it follows that for $Z \xrightarrow{[a,b]} B_2$ it holds $b - a < \mathrm{cont}_S(B_2)$. However, as shown previously this implies that the network can no longer be DC. $\square$

From the previous observations, we can derive important relationships between lowerGuard(C), upperGuard(C) and cont(C) values:

Lemma 4. *Let S be a dynamically controllable STNPSU, Z be its initial timepoint and C be any other timepoint. If T is the network derived from S by restricting upper bound v of the distance $Z \xrightarrow{[u,v]} C$ between Z and C to v^*, with* $\mathrm{upperGuard}_S(C) \leq v^* \leq v$, *in T it holds*

$$\mathrm{lowerGuard}_T(C) = \min \{\mathrm{lowerGuard}_S(C); v^* - \mathrm{cont}_S(C)\}$$

Lemma 5. *Let S be a dynamically controllable STNPSU, Z be its initial timepoint and C be any other timepoint. If T is the network derived from S by restricting the lower bound u of the distance $Z \xrightarrow{[u,v]} C$ between Z and C to u^*, with $u \leq u^* \leq$ $\mathrm{lowerGuard}_S(C)$, in T it holds*

$$\mathrm{upperGuard}_T(C) = \max \{\mathrm{upperGuard}_S(C); u^* + \mathrm{cont}_S(C)\}$$

Proof. The proofs of Lemmas 4 and 5 are very similar. For the sake of brevity, we only show that lemma 4 holds.

First, let us assume that $\mathrm{lowerGuard}_T(C) > v^* - \mathrm{cont}_S(C)$. That is, by Def. 2 and Lemma 2 it holds that u can be increased to $u^* = \mathrm{lowerGuard}_T(C) > v^* - \mathrm{cont}_S(C)$. However, then by Lemma 3 the resulting network cannot be DC.

Second, let us assume that u is increased to $u^* = v^* - \mathrm{cont}_S(C) \leq \mathrm{lowerGuard}_S(C)$ in T and that the resulting network is not DC. This implies that there exists a negative semi-reducible cycle

$$Z \xrightarrow{\alpha_0} E_1 \xrightarrow{\alpha_1} E_2 \ldots \xrightarrow{\alpha_{l-1}} E_l \xrightarrow{\alpha_l} C \xrightarrow{-(v^* - \mathrm{cont}_S(C))} Z$$

in the distance graph $\mathcal{D}_T$ of T such that $\sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i} - (v^* - \mathrm{cont}_S(C)) < 0$, i.e., $\mathrm{cont}_S(C) < v^* - \sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i}$. Moreover, it holds that $v^* \leq v \leq \sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i}$ and thus $\mathrm{cont}_S(C) < v^* - \sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i} \leq 0$ which contradicts the basic property that $\mathrm{cont}_S(C) \geq 0$.

Third, let us assume that u is increased to $u^* = \mathrm{lowerGuard}_S(C) \leq v^* - \mathrm{cont}_S(C)$ and that the resulting network is not DC. This again implies that there exists a negative semi-reducible cycle

$$Z \xrightarrow{\alpha_0} E_1 \xrightarrow{\alpha_1} E_2 \ldots \xrightarrow{\alpha_{l-1}} E_l \xrightarrow{\alpha_l} C \xrightarrow{-\mathrm{lowerGuard}_S(C)} Z$$

in the distance graph $\mathcal{D}_T$ of T such that $\sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i} - \mathrm{lowerGuard}_S(C) < 0$, i.e., $\sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i} <$ $\mathrm{lowerGuard}_S(C)$. Thus it also holds $\sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i} < \mathrm{lowerGuard}_S(C) \leq v^* - \mathrm{cont}_S(C) \leq v^* \leq v$, i.e., $\sum_{i \in \{1,\ldots,l\}} \tilde{\alpha_i} < v$ which contradicts the basic assumption that v has been restricted to v^*. $\square$

3.4. Overall Temporal Properties of a Process

The previous results give rise to the following theorem that enables a complete description of the overall temporal properties of a process.

Theorem 2 (Overall Temporal Properties of a Process)**.** *Considering a process P and the corresponding STNPSU S, let Z and E be the single start and single end timepoints of S. Then: The* overall temporal properties *of P can be described by a* guarded range with contingency $[[x, x'][y', y]] \updownarrow c$, *where*

- *x and y are the bounds of the requirement link $Z \xrightarrow{[x,y]} E$ between initial timepoint Z and ending timepoint E in S, as derived by the DC checking algorithm,*
- *$x' = \mathrm{lowerGuard}_S(E)$ and $y' = \mathrm{upperGuard}_S(E)$, and*
- *$c = \mathrm{cont}_S(E)$.*

Proof. Defs. 1 and 2 show how to use the values of $\text{lowerGuard}_S(E) = x'$ and $\text{upperGuard}_S(E) = y'$ to specify the possible restrictions regarding the lower and upper bounds of the duration range $[x, y]$ of a process (i.e., its minimum and maximum duration). This way, we can fully represent the possible duration ranges of the process as a guarded range $[\![x, x'][y', y]\!]$. Moreover, Lemmas 3–5 show how to use the path contingency span $\text{cont}_S(E) = c$ in order to ensure that any possible restriction of the duration range $[\![x, x'][y', y]\!]\updownarrow c$ of the process preserves its DC. □ □

Based on Theorem 2, it becomes possible to represent the overall temporal properties of a process using a single guarded range with contingency, as illustrated by Example 7.

Example 7. First, consider process P_1 as depicted in Fig. 1 together with the corresponding STNPSU shown in Fig. 3. The overall temporal properties of this process may be described by guarded range with contingency $[\![5, 13][11, 19]\!]\updownarrow 2$. Since the contingency span of this process corresponds to 2, it is possible to restrict the overall duration range of the process to $[13, 15]$ or $[9, 11]$, while still preserving its DC. In turn, the overall temporal properties of process P_2 (cf. Figs. 1 and 3) can be described by a guarded range with contingency $[\![5, 10][14, 19]\!]\updownarrow 6$. For example, the duration range of the process, therefore, can be restricted to $[6, 14]$, $[10, 17]$, or $[8, 14]$. However, due to the required contingency span of 6, for example, it must not be restricted to $[10, 14]$, or $[10, 15]$.

Such kind of compact representation of the overall temporal properties of a process schema is crucial for being able to reuse it as part of a modularized process. In particular, when adding a subprocess task to a process schema, a duration range for the respective task must be specified. Based on the guarded range with contingency determined for the subprocess it is now possible to determine a proper duration range for the respective subprocess task. This duration range ensures that, without having to reanalyze the subprocess schema, any restriction of the duration of the subprocess task will be made in such a way that the respective subprocess remains dynamically controllable.

4. DC-Checking of Modularized Time-Aware Processes

As shown in the previous section, for each time-aware process, it is possible to derive a *guarded range with contingency* that fully describes the overall temporal properties of the process. In particular, this guarded range with contingency specifies the possible durations of the process as well as the permissible restrictions that may be applied to the duration range of the process without violating its DC. In this section we show how this knowledge may be utilized for enabling a sophisticated support of modularized time-aware processes in a PAIS.

In a PAIS, the available process schemas are generally stored in a central process model repository [26]. Based on the results presented in Sect. 3, it now becomes possible to enhance the information about the process schemas in such a repository with the overall temporal properties of the process schema represented as a guarded range with contingency.

Such information can then be utilized when re-using a process schema as part of a modularized time-aware process. In particular, during design time a process designer may select a process schema from the repository to be used as a subprocess task. Similar to an atomic task, the designer then has to configure the subprocess task within the process schema; i.e., he must specify the duration range of the particular subprocess task. In order to ensure the executability of the modularized process the designer must guarantee that the duration range set for the subprocess task is compliant with the overall temporal properties of the (sub-)process schema. In this context, the repository information about the overall temporal properties of the (sub-)process schema may be used to support the process designer in choosing a proper duration range for the respective subprocess task. In other words, the designer must select a guarded range as duration range of the subprocess task, which satisfies the guards as well as the contingency of the guarded range with contingency representing the overall temporal properties of the (sub-)process schema as stored in the repository.

In general, the duration range $[\![x, x'][y', y]\!]$ of a subprocess task needs to be selected with respect to the overall temporal properties of the respective (sub-)process schema $[\![u, u'][v', v]\!]\updownarrow c$ such that $u \le x \le x' \le u'$

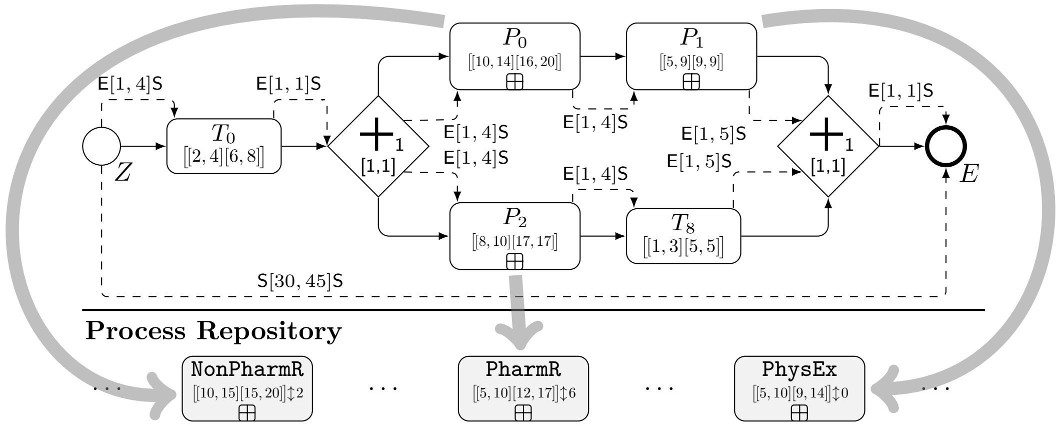


Figure 5: Modularized process.

and $v \geq y \geq y' \geq v'$ hold. Moreover, if $c > 0$ holds, $y' - x' \geq c$ must hold as well. When observing these constraints, it is guaranteed that, during the execution of a subprocess task of a modularized process, the respective subprocess instance may be completed without violating any of its temporal constraints (i.e., the subprocess is DC).

Example 8. Fig. 5 depicts the modularized process from Fig. 1 where proper duration ranges have been selected for the three subprocess tasks P_0, P_1 and P_2, which are related to (sub-)process schemas `NonPharmR`, `PhysEx` and `PharmR`. For example, for subprocess task P_0, duration range $[\![10, 14][16, 20]\!]$ is used. This range has the same outer bounds as the overall temporal properties of the respective process schema, i.e., $[\![10, 15][15, 20]\!]\updownarrow 2$. Moreover, the lower and upper guard of the duration range ensure that the guards as well as contingency value determined for the process schema are observed. In turn, for subprocess task P_1 the designer decides to further restrict the upper bound of the duration range to 9 (thus also decreasing the lower guard to 9). Note that this still guarantees the DC of subprocess schema `PhysEx` as it complies with the respective guards and contingency. Finally, for subprocess P_2, the designer increased the lower bound to 8 and the upper guard to 17, thus providing a possible contingency of 7 instead of the required contingency of 6.

After completing the design of the modularized process schema, the dynamic controllability of the parent process schema itself needs to be verified. Then, the overall temporal properties of the modularized process schema may be determined based of the approach presented in Sect. 3.

Finally, the modularized process itself may be added to the process repository. It may then be reused as a subprocess in the context of another modularized process. This enables the definition of hierarchically structured modularized time-aware process schemas comprising multiple levels.

5. Proof of Concept

The presented approach was implemented as a proof-of-concept prototype in the ATAPIS Toolset [14, 15]. This prototype enables users to create time-aware process schemas and to automatically transform them to a corresponding STNPSU. The STNPSU can then be checked for dynamic controllability. Moreover, the overall temporal properties of the process can be determined.

The screenshot from Fig. 6 shows the ATAPIS Toolset[1]: at the top, the process schema from Fig. 1b is shown. At the bottom, the automatically generated STNPSU and its minimal network are depicted.

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

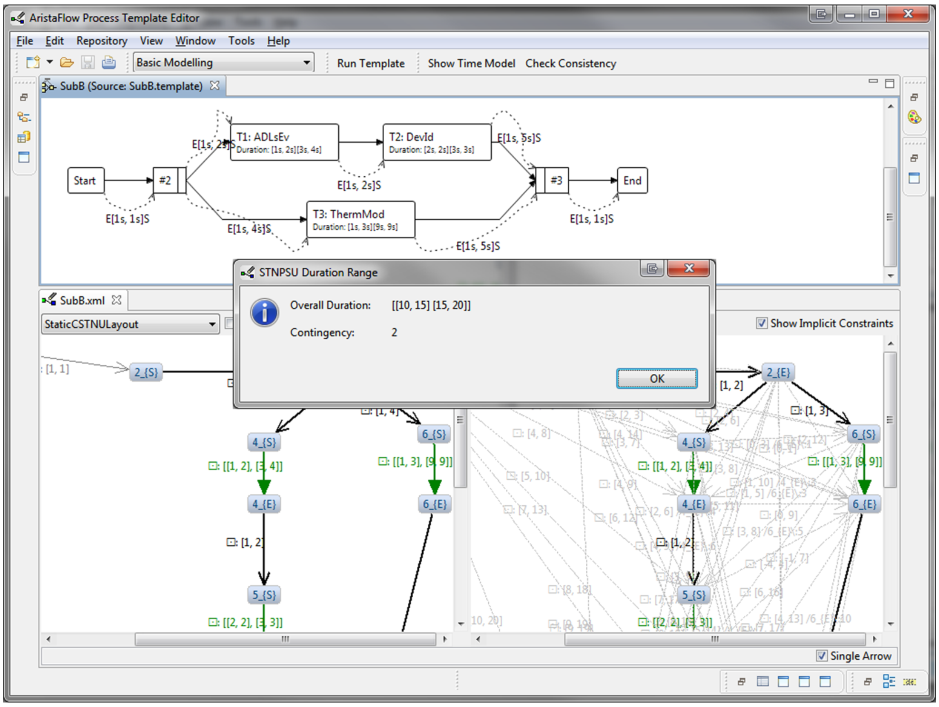


Figure 6: Determining the Overall Temporal Properties of a Process in the ATAPIS Toolset.

Finally, the dialog in the middle shows the overall temporal properties of the process schema which have been determined based on the STNPSU.

Moreover, using the ATAPIS prototype it becomes possible to create modularized time-aware processes and to assign a proper duration range to each subprocess task based on the overall temporal properties of the respective (sub-)process schema. The resulting modularized time-aware process schema can then be checked for dynamic controllability and its overall temporal properties be determined. It is then possible to reuse this modularized time-aware process schema for a subprocess task in another modularized process.

First simulations based on the ATAPIS prototype show a significantly improved performance of our modularization-based approach compared to the "classical approach" where each subprocess task has to be replaced by it respective (temporal) components.

Overall, the prototype demonstrates the applicability of our approach.

6. Conclusions

Time and modular design constitute two fundamental aspects for properly supporting business processes by PAIS. So far, these aspects have only been considered in isolation, although the overall temporal behaviour of a (sub-)process significantly differs from the one of simple tasks. This paper closes this gap by considering modularization and time-awareness of processes in conjunction with each other. In particular, we propose a novel approach for determining and representing the overall temporal behavior of a process, called *guarded range with contingency*. Using this representation, we can specify the possible durations of a (sub-)process as well as any permissible restriction that may be applied to it, while still ensuring the executability of the process. Moreover, we show how this may be used in the context of process repositories and multilayered process hierarchies. Finally, the presented approach was fully implemented as part of our ATAPIS Toolset demonstrating its feasibility.

We are currently extending STNPSU to consider conditional aspects as well. In this context, we will also revisit the presented approach. In future work, we want to study the integration of (modularized) time-aware processes in PAISs, specifically focusing on aspects like scalability and usability. In this context, the presented approach will play a crucial role enabling the efficient and modular design of time-aware processes. Finally, we would like to explore the concept of modularization in the context of temporal networks in order to improve the performance of controllability checking of such network.

References

[1] Combi, C., Gambini, M., Migliorini, S., Posenato, R.: Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. IEEE T. Systems, Man, and Cybernetics: Systems 44(9), 1182–1203 (Sep 2014)

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

[3] Combi, C., Hunsberger, L., Posenato, R.: An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In: Filipe, J., Fred, A.L.N. (eds.) ICAART 2013 - Proc of the 5th International Conference on Agents and Artificial Intelligence, Volume 2. pp. 144–156. SciTePress (Feb 2013)

[4] Combi, C., Posenato, R.: Controllability in temporal conceptual workflow schemata. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) Business Process Management, BPM 2009, Proc, LNCS, vol. 5701, pp. 64–79. Springer (Sep 2009)

[5] Combi, C., Posenato, R.: Towards temporal controllabilities for workflow schemata. In: Markey, N., Wijsen, J. (eds.) TIME 2010 - 17th Intern. Symp. on Temporal Representation and Reasoning. pp. 129–136. IEEE Computer Society (Sep 2010)

[6] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press (2001)

[7] Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence 49(1-3), 61–95 (1991)

[8] Eder, J., Euthimios, P., Pozewaunig, H., Rabinovich, M.: Time management in workflow systems. In: Proc. BIS'99. pp. 265–280. Springer (1999)

[9] Eder, J., Gruber, W., Panagos, E.: Temporal modeling of workflows with conditional execution paths. In: Proc. DEXA'00. pp. 243–253. Springer (Sep 2000)

[10] Eder, J., Panagos, E., Rabinovich, M.: Workflow time management revisited. In: Seminal Contributions to Information Systems Engineering, pp. 207–213 (2013)

[11] Hochberg, M.C., et al.: American college of rheumatology 2012 recommendations for the use of nonpharmacologic and pharmacologic therapies in osteoarthritis of the hand, hip, and knee. Arthritis Care & Research 64(4), 465–474 (2012)

[12] Lanz, A., Posenato, R., Combi, C., Reichert, M.: Controllability of time-aware processes at run time. In: Meersman, R., Panetto, H., Dillon, T.S., Eder, J., et al. (eds.) On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE. LNCS, vol. 8185, pp. 39–56. Springer (Sep 2013)

[13] Lanz, A., Posenato, R., Combi, C., Reichert, M.: Simple temporal networks with partially shrinkable uncertainty. In: Loiseau, S., Filipe, J., Duval, B., van den Herik, H.J. (eds.) ICAART 2015 - Proc of the International Conference on Agents and Artificial Intelligence, Volume 2. pp. 370–381. SciTePress (2015)

[14] Lanz, A., Reichert, M.: Dealing with changes of time-aware processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) Proc BPM'14. LNCS, vol. 8659, pp. 217–233. Springer (2014)

[15] Lanz, A., Reichert, M.: Enabling time-aware process support with the atapis toolset. In: Proc. BPM'14 Demo Track (2014)

[16] Lanz, A., Reichert, M., Weber, B.: A formal semantics of time patterns for process-aware information systems. Tech. Rep. UIB-2013-02, University of Ulm (2013)

[17] Lanz, A., Weber, B., Reichert, M.: Workflow time patterns for process-aware information systems. In: Proc of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010. LNBIP, vol. 50, pp. 94–107. Springer (2010)

[18] Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requirements Engineering 19(2), 113–141 (2014)

[19] Marjanovic, O., Orlowska, M.E.: On modeling and verification of temporal constraints in production workflows. Knowl. and Inf. Syst. 1(2), 157–192 (1999)

[20] Morris, P.: A structural characterization of temporal dynamic controllability. In: Benhamou, F. (ed.) Intl Conf on Principles and Practices of Constraint Programming (CP'06). pp. 375–389. Springer (2006)

[21] Morris, P.H., Muscettola, N.: Temporal dynamic controllability revisited. In: National Conf on Artificial Intelligence (AAAI'05). pp. 1193–1198 (2005)

[22] Morris, P.H., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: Intl Joint Conf on Artificial Intelligence (IJCAI'01). pp. 494–502. Morgan Kaufmann (2001)

[23] Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer (2012)

[24] Reijers, H.A.: Design and Control of Workflow Processes. Springer (2003)

[25] Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering 68(9), 793–818 (Sep 2009)

[26] Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring large process model repositories. Computers in Industry 62(5), 467 – 486 (2011)