Introducing Agile Controllability in Temporal Business Processes

Roberto Posenato[1(✉)], Marco Franceschetti[2], Carlo Combi[1], and Johann Eder[3]

[1] Department of Computer Science, University of Verona, Verona, Italy
{roberto.posenato,carlo.combi}@univr.it

[2] Institute of Computer Science, University of St. Gallen, St. Gallen, Switzerland
marco.franceschetti@unisg.ch

[3] Department Informatics-Systems, University of Klagenfurt, Klagenfurt, Austria
johann.eder@aau.at

Abstract. Dynamic controllability is currently regarded as the most adequate notion for checking the temporal correctness of business processes with temporal constraints when a process model includes uncontrollable activities whose duration is revealed at the time of activity completion. However, dynamic controllability cannot take advantage when an actual duration is revealed earlier, leading to unnecessary strict checks for temporal correctness. We propose a novel notion of *agile controllability*, which takes into account that uncontrollable durations are revealed earlier and that in a viable execution strategy, a time point may depend on time points whose value is known earlier. We formalize the notion of agile controllability and present an effective checking procedure evaluated by a software implementation within a publicly available modeling and checking software tool.

Keywords: business processes · temporal constraints · agile controllability · oracles

1 Introduction

Temporal business process models are subject to requirements stating constraints on the durations and execution times of activities [12]. Durations state the minimum or maximum permissible duration of activities, as well as whether an agent may be responsible for deciding an activity duration within a predetermined interval [17] or may only be able to observe the actual duration of an activity that is controlled elsewhere. Temporal constraints restrict the permissible time windows between the occurrence of events –start and end times of activities– or temporal parameters [4].

Temporal process models are typically checked at design-time for temporal correctness to ensure that compliance with these temporal requirements can be achieved at run time, i.e., no temporal constraints violated [6]. For processes entirely under the control of an agent, the *satisfiability* of temporal constraints is accepted as an adequate notion for temporal correctness [3]. However, for processes in which some activity durations, known as *contingent* durations, are not controllable by the agent, satisfiability is insufficient [2]. Here, more sophisticated notions such as *dynamic controllability* are required [13,17]. These notions account for the fact that the actual duration of a contingent activity is only known when it completes: for instance, the actual duration of a Web service call is uncontrollable and known only when a value is returned, although Service Level Agreements guarantee that this will happen within a specified interval.

Dynamic controllability requires that a viable execution strategy for a process with contingent activities exists, in which later (greater) timepoints may depend on earlier (smaller) time points but not vice versa. However, the assumption that a contingent duration is only revealed at the time of activity completion is frequently too restrictive. Indeed, although an activity duration might be uncontrollable, the exact duration could be known *before* the activity completes. For example, the shipment of an order whose duration is specified to be within an uncontrollable (by the buyer) range of 6 to 10 days at the time of order placement; however, within 2 days after placing an order, the actual duration of the shipment is communicated - e.g., the shipment will take precisely seven days. In such cases, knowing the exact duration in advance allows for more flexible process scheduling since uncertainty for the agent is reduced earlier, and the agent may use this information for scheduling activities that are after the information time point but before the receipt of the ordered item. In the example, the buyer can schedule an activity that has to be executed exactly 2 days *before* the delivery since she already knows the delivery date.

Current state-of-the-art procedures for checking the dynamic controllability of a process model are based on mapping the model into a Simple Temporal Network with Uncertainty (STNU) [2,9]. However, the STNU lacks constructs for modeling contingent durations that are revealed in advance. Hence, current STNU procedures return potentially overly restrictive results for process models where contingent durations are known earlier. Specifically, these models are classified as not dynamically controllable, while in reality, they can be scheduled to avoid constraint violations thanks to duration information communicated in advance. This misclassification leads to unnecessary and costly model redesign. This shortcoming calls for a new representation of process models where contingent durations are revealed before their completion, allowing an effective design-time check for temporal correctness–the challenge we take on here.

The contribution of this paper is a conservative extension of the STNU that overcomes the limitation of not being able to represent information on contingent durations in advance. We introduce the concept of *contingent oracle* to represent the point in time in which a contingent duration is revealed, allowing checking the temporal correctness of such process models where contingent durations are revealed in advance. We define *agile controllability* as the temporal correctness of an STNU with contingent oracle nodes where an execution strategy is viable if timepoints may depend only on time points, which are known earlier. We discuss an algorithm to check agile controllability and, as proof of concept, an open-source implementation.

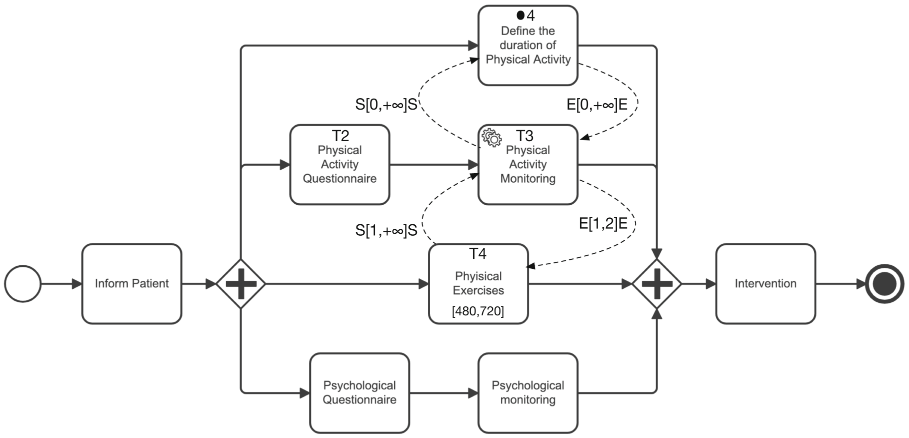

●4
Define the duration of Physical Activity
S[0,+∞]S
E[0,+∞]E
T2
Physical Activity Questionnaire
T3
Physical Activity Monitoring
S[1,+∞]S
E[1,2]E
T4
Phyisical Exercises
[480,720]
Inform Patient
Intervention
Psychological Questionnaire
Psychological monitoring

Fig. 1. BPMN diagram for the process for the patient pre-intervention period.

2 Motivating Example

Figure 1 represents a motivating example, which we will briefly discuss here. The example refers to managing patients' activities before a surgical intervention. It is becoming widely acknowledged that patients reaching some planned intervention in an (as much as possible) good health state have a better recovery [11].

The process in the figure starts with the usual information to patients, who must be aware of the following activities. Then, different threads of activities are initiated. Two threads are related to the Physical and Psychological parts, respectively. The proposed physical exercises depend on and can be refined with respect to the results of different questionnaires the patient has to answer. After the end of these activities, the intervention is performed. Focusing on the process fragment related to the physical exercise activity and the related monitoring, these activities are connected through some temporal constraints, representing the allowed delays between their start and endpoints, respectively. Moreover, activities are enriched with their allowed time duration. In the figure, only the temporal duration and the temporal constraints relevant to our discussion are reported. The notation for task durations and inter-task constraints is relatively standard in the literature [1]. Tasks have a duration attribute represented as a range $[x, y]$, with $0 < x \leq y < \infty$, where x/y is the minimum/maximum allowed time span for an activity to go from state "started" to "completed" [12]. Inter-task constraints limit the time distance between the starting/ending instants of two tasks and have the form $I_S[u, v]I_F$, where I_S is the starting (S)/ending (E) instant of the first task, while I_F is the starting/ending instant of the second one [1].

Task T4 "Physical Exercises" (PE) activity has to be performed for a period from 20 to 30 days (480 to 720 h). Task T3 "Physical Activity Monitoring" (PAM) has to start (at least) 1 h after the start of physical exercises and end

1 to 2 h before the exercises end to avoid noisy information during the initial and final phases of physical activities. As the contingent duration of PE is not determined by the system, it can only be observed when the task ends; hence PAM cannot be set to end according to the required time distance before PE. This observation holds both if we consider PAM a contingent task, having a duration only observable by the system, and if we consider it a *controllable* task, for which the system can set the duration. Indeed, in any case, the ending instant of PAM, even if controllable, should be set with respect to the ending instant of PE, which has to occur afterward. Thus, the PAM ending instant cannot be adequately set unless there is some kind of early specification/knowledge about the (contingent) duration of PE.

Let us now explicitly specify that the duration of PE is set by the activity O4 “Define the duration of Physical Activity” (DDPA). Let us also assume, for the sake of simplicity, that the system can decide when the PAM has to end. From this perspective, how do we derive when it is required to execute DDPA to make the overall process model controllable? It is straightforward to verify that if the system knows the overall duration of PE at least two hours before the effective end of PE, then the end of PAM can be set, satisfying the given constraints.

To ensure that all temporal constraints specified in the process can be satisfied in any execution, no matter how long contingent durations take, we want to be able to check (at design time) the dynamic controllability of the process model. This check is typically done by mapping the model into a proper temporal constraint network (like Simple Temporal Network with Uncertainty).

3 STNUs and Oracles

The Simple Temporal Network with Uncertainty (STNU) [13] is a data structure that models temporal problems, such as time-constrained process models, in which the execution of some events cannot be controlled. The STNU comprises a set of timepoints and a set of temporal constraints. The timepoint set is partitioned into controllable (executable) timepoints and uncontrollable (contingent) ones; the constraint set is partitioned into regular and contingent ones.

To the best of our knowledge, in the current temporal business process models and/or in the related temporal constraint networks, it is not possible to represent the possibility that a contingent duration is known before the occurrence of the corresponding contingent timepoint. Here, we consider the following issues:

– how to extend STNUs to represent contingent temporal constraints having their duration acquired/known possibly before their occurrences?
– How to derive when such extended STNUs are controllable, i.e., how early do we need to acquire/know the duration of some contingent link?

As the STNU does not allow decoupling the value of a contingent duration and the time of occurrence of the associated timepoint, we introduce a new kind of timepoint called *(contingent) oracle*. An oracle O_C is a timepoint associated bi-univocally with a contingent link (A, C). When O_C is executed, it reveals the duration of the associated contingent link. In other words, O_C can reveal the duration of the contingent link before the contingent timepoint C occurs. We extend the formal definition of an STNU in [7] with oracles as follows:

Definition 1 (STNU with Oracles). *An STNU with Oracles (STNUO) is a tuple* $(\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$*, where:*

- $\mathcal{T}$ *is a finite, non-empty set of real-valued variables called timepoints.* $\mathcal{T}$ *is partitioned into* $\mathcal{T}_X$*, the set of executable timepoints and* $\mathcal{T}_C$*, the set of contingent timepoints.* $\mathcal{T}_O \subseteq \mathcal{T}_X$*, is the set of oracle timepoints.*
- $\mathcal{C}$ *is a set of binary (ordinary) constraints, each of the form* $Y - X \leq \delta$ *for some* $X, Y \in \mathcal{T}$ *and* $\delta \in \mathbf{R}$.
- $\mathcal{L}$ *is a set of contingent links, each of the form* (A, x, y, C)*, where* $A \in \mathcal{T}_X, C \in \mathcal{T}_C$ *and* $0 < x < y < \infty$*. A is called the* activation *timepoint; C* contingent *timepoint. If* (A_1, x_1, y_1, C_1) *and* (A_2, x_2, y_2, C_2) *are distinct contingent links, then* $C_1 \neq C_2$.
- $\mathcal{O}\colon \mathcal{T}_O \to \mathcal{T}_C$ *is an injective function that associates an oracle with its corresponding contingent timepoint. For each pair* (O_C, C) *such that* $\mathcal{O}(O_C) = C$*, there must exist the constraint* $O_C - C \leq 0$ *in* $\mathcal{C}$.

We say that a controller *executes an STNUO* when it schedules its timepoints, i.e., it assigns a value to each timepoint (for contingent ones, the assignment is derived once the relative contingent duration is revealed).

An important property of the STNU is the *dynamic controllability*. An STNU is *dynamically controllable* (DC) if there exists a dynamic execution strategy (called *viable execution strategy*) that assigns the timepoints with the guarantee that all constraints will be satisfied, irrespectively from the values (within the specified bounds) of the contingent durations [7].

In the literature, there are several proposals to check dynamic controllability based on constraint propagation techniques, defined on the *distance graph* corresponding to a given STNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ [8,13]. A distance graph associated with an STNUO is the graph $(\mathcal{T}, \mathcal{E} = \mathcal{E}_\mathcal{C} \cup \mathcal{E}_\mathcal{L})$, where the timepoints in $\mathcal{T}$ serve as the graph's nodes and the constraints in $\mathcal{C}$ and $\mathcal{L}$ correspond to labeled, directed edges in $\mathcal{E}$. In particular: $\mathcal{E}_\mathcal{C} = \{X \xrightarrow{\delta} Y \mid (Y - X \leq \delta) \in \mathcal{C}\}$ while for $\mathcal{E}_\mathcal{L}$ there are two edges for each $(A, x, y, C) \in \mathcal{L}$: the *lower-case* (LC) edge $A \xrightarrow{c:x} C$ in $\mathcal{E}_\mathcal{L}$ represents the *uncontrollable possibility* that the duration $C - A$ may be as low as x; the *upper-case* (UC) edge $C \xrightarrow{C:-y} A$ in $\mathcal{E}_\mathcal{L}$ represents the *uncontrollable possibility* that the duration $C - A$ may be as high as y.

A constraint propagation technique consists of applying constraint propagation rules in the distance graph to make explicit derived constraints from the existing ones in the STNU. The process terminates when either reaching quiescence, i.e., no new constraints can be derived (the network is dynamically controllable), or a negative circuit is found (the network is not dynamically controllable).

Now, let us consider the more structured case of Fig. 1. Using some transformation rules like the ones presented in [16], it is possible to represent the process as an STNUO instance. Figure 2 represents an excerpt of the STNUO, focused only on three activities of the process: T2, T3, O4, and T4. Each activity is

$$\begin{tikzcd}
A_2 \arrow[r, shift left, "2"] \arrow[d, "0"] & B_2 \arrow[l, shift left, "-1"] & A_3 \arrow[l, "0"'] \arrow[d, "0"] & O_4 \arrow[l, "0"'] & B_3 \arrow[d, shift left, "2"] & \Omega \arrow[l, "0"'] \arrow[dl, bend left, "0"] \\
Z & & A_4 \arrow[ll, "0"'] \arrow[rr, shift left, "{720, c_4{:}480}"] & & C_4 \arrow[ll, shift left, "{-480, C_4{:}-720}"] \arrow[ul, "0"] \arrow[u, shift left, "-1"] &
\end{tikzcd}$$

A_2, B_2 represent the starting/ending event of T2 'Physical Activity Questionnaire'
A_3, B_3 represent the starting/ending event of T3 'Physical Activity Monitoring'
A_4, C_4 represent the starting/ending event of T4 'Physical Exercises'
O_4 represents the end of O4 'Define the duration of Physical Activity'

Fig. 2. STNU distance graph representing a possible conversion of a part of the BPMN model in Fig. 1

represented by two timepoints, one representing the starting instant, the other the ending one, and one or two constraints on its duration. According to the kind of duration (controllable or not), the duration constraints can be ordinary or contingent.

Let us consider each activity.

- T2 is a controllable activity. The execution agent can fix its duration. Therefore, it is represented by the two timepoints A_2 and B_2 and by two ordinary constraints between A_2 and B_2 that impose that the duration must be in the range $[1, 2]$. Without loss of generality, we assume that the time granularity is *hours* and that all constraint values are integers.
- T4 is a contingent activity since it is possible to fix when it must start, but given a possible duration, the end of the activity can only be observed when it occurs. Therefore, it is represented as two timepoints A_4 and C_4, and by two constraints representing the upper-case and lower-case of the contingent link associated with the activity. The duration of such an activity must be in the range of 20 to 30 days, i.e., $[480, 720]$ using the time unit of hours.
- T3 is an activity that depends on T4. Therefore, its timepoints are A_3 and B_3, which are constrained to be just after A_4 and one hour before C_4 at least, respectively.
- Last, O4 is an activity represented just by one timepoint, O_4. We assume that such activity is instantaneous since it represents just the instant in which the duration of T4 is decided. In the original BPMN process, such activity is constrained to be after the start of T3, which must start after T4. O4 reveals a piece of information that can be crucial to executing the process without violating any temporal constraint. Thus, it is the oracle associated with the activity T4.

In [15], we proposed the definition of oracle dependency, and we extended the ones of viable execution strategy and dynamic execution strategy when oracle timepoints are present. Here, we recall the definition of agile controllability: *an STNU with contingent oracles (STNUO) is agilely controllable if it admits a viable dynamic execution strategy with contingent oracles. We refer to agile controllability (AC) as the property of being agilely controllable.*

Table 1. Propagation rules for checking Agile Controllability of STNUO.

Rule	**Conditions**	**Pre-existing** and **generated** edges
No Case (NC)		$\begin{tikzcd} W & Y \arrow[l, "v"'] & X \arrow[l, "u"'] \arrow[ll, bend left, "u+v"] \end{tikzcd}$
Cross Case (CC)	$D \not\equiv C$; $v < 0$	$\begin{tikzcd} X & C \arrow[l, "D{:}v"'] & A \arrow[l, "c{:}u"'] \arrow[ll, bend left, "D{:}u+v"] \end{tikzcd}$
Label Removal (LR)	$v \geq -x$	$\begin{tikzcd} C & A \arrow[l, "c{:}x"'] & X \arrow[l, "C{:}v"', "v"] \end{tikzcd}$
New Lower Case (nLC)	$-u \leq 0$; O_C does not exists or $v - u \geq y - x$	$\begin{tikzcd} X \arrow[r, shift left, "v"] & C \arrow[l, shift left, "-u"] \arrow[r, shift left, "C{:}-y"] & A \arrow[l, shift left, "c{:}x"] \arrow[ll, bend left, "x-u"] \end{tikzcd}$
New Upper Case (nUC)	O_C does not exist or $v - u \geq y - x$	$\begin{tikzcd} X \arrow[r, shift left, "v"] \arrow[rr, bend right, "C{:}v-y"'] & C \arrow[l, shift left, "-u"] \arrow[r, shift left, "C{:}-y"] & A \arrow[l, shift left, "c{:}x"] \end{tikzcd}$
UC*	$Y \not\equiv C$	$\begin{tikzcd} X \arrow[r, "u"] \arrow[rr, bend right, "C{:}x+v"'] & Y \arrow[r, "C{:}v"] & A \end{tikzcd}$
Oracle (ORC)	X is **not** a contingent node; O_C exists; $v - u < y - x$.	$\begin{tikzcd} X \arrow[r, shift left, "v"] \arrow[rr, bend left, "v-x"] \arrow[dr, "0"'] & C \arrow[l, shift left, "-u"] \arrow[r, shift left, "C{:}-y"] \arrow[d, "-u"] & A \arrow[l, shift left, "c{:}x"] \arrow[dl, "x-u"] \arrow[ll, bend left=80, "y-u"] \\ & O_C & \end{tikzcd}$

Two different issues arise with the agile controllability of temporal BPM models: (i) checking whether a given temporal BPM model with oracles is agilely controllable and (ii) determining a schedule for the given model. In the following, we discuss how to check the agile controllability of an STNUO.

4 Checking Agile Controllability

One possibility for checking the agile controllability is to exploit and expand the approach proposed by Morris-Muscettola (MM) for checking the dynamic controllability of STNUs, based on the constraint propagation [13].

In Table 1, we propose the set of MM rules modified for checking the agile controllability. In particular, the first three rules in the table are original, while rules nLC, nUC, and UC* are a restriction of the corresponding original rules LC and UC in [13], while ORC rule is a new one specific for considering oracles.

The rules were designed assuming the following properties:

- *instantaneous reaction semantics;* i.e., *"when the environment decides the duration of a contingent link, the agent can react at the same time executing one or more other timepoints"*.
- *early-execution strategy;* i.e., timepoints are executed at the first possible instant. Rules are able to determine such an instant, but they do not determine the latest possible execution instant for each timepoint.

The MM DC-checking algorithm applies the rules in at most n^2 rounds, with a cost of $O(n^3)$ each round, for a global execution time of $O(n^5)$, where n is the number of network nodes.

The original rules allow us to determine a correct lower bound to the execution time of an executable X with respect to all possible durations of a contingent link (A, x, y, C). When an executable X has to be "strictly" scheduled with respect to the duration of a contingent link (A, x, y, C) with oracle, LC, and UC determine a *false* negative cycle. For example, if there is a pattern like $X \xrightleftharpoons[-1]{2} C \xrightleftharpoons[c:1]{C:-10} A$, applying LC and UC, the resulting constraints would be $\mathrm{X} \xrightleftharpoons[0]{C:-8} \mathrm{A}$ that represent a negative cycle. Since there are oracles in STNUOs that can help to schedule such X, it is necessary to limit the application of the original LC and UC rules (when oracles are available) to avoid a network being classified as not controllable while it is agilely controllable.

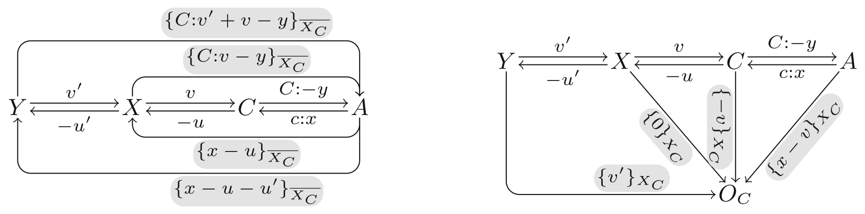
A timepoint X is "strictly" scheduled with respect to a contingent link (A, x, y, C) when: (i) X must occur before C (there is an edge between C and X with a negative value), (ii) the distance range between X and C is smaller than the contingent allowed duration. In other words, if there is a pattern like $X \xrightleftharpoons[-u]{v} C \xrightleftharpoons[c:x]{C:-y} A$, where $-u \leq 0$, $v - u < y - x$, and there is an oracle O_C for contingent link (A, x, y, C).

Therefore, the nLC rule replaces LC. It must be applied only when there is no oracle for the contingent link, or the span $v - u$ is greater than or equal to the span $y - x$. Indeed, when $v - u \geq y - x$, X can be scheduled for any possible duration of the contingent link without the necessity of an oracle.

nUC and UC* rules replace UC. In the case of the propagation of the upper-case labeled value of a contingent link, rule nUC must be considered, and it must be applied only when the contingent link (A, x, y, C) does not have its oracle node O_C, or when $v - u \geq y - x$. When an upper-case labeled value has been propagated, then the following propagations must be done by using rule UC* (in [13] such a rule was included in the UC rule).

When nLC and nUC cannot be applied, it is necessary to verify if it is possible to exploit the presence of a possible oracle associated with the contingent link to check whether X can be scheduled correctly. A possible rule that properly constrains an oracle O_C of a contingent C and a node X that has to be "strictly" scheduled is rule ORC in Table 1. Given a pattern $X \xrightleftharpoons[-u]{v} C \xrightleftharpoons[c:x]{C:-y} A$, where $-u \leq 0$, $v - u < y - x$, and an oracle O_C for contingent link (A, x, y, C), it is necessary to require that O_C must before both C and X with a proper distance for verifying whether it is possible to schedule X correctly w.r.t. any possible duration of contingent link. Therefore, constraints $C \xrightarrow{-u} O_C$ and $X \xrightarrow{0} O_C$ are added. Constraints $A \xrightarrow{x-u} O_C$, $A \xrightarrow{y-u} X$, and $X \xrightarrow{v-x} A$ are then determined by applying the NC rule; we propose to add such values directly to understand the rule better.

The soundness of the rules in Table 1 and the completeness of the general algorithm have been presented in [15].


$\{C{:}v' + v - y\}_{\overline{X_C}}$
$\{C{:}v - y\}_{\overline{X_C}}$
Y
v'
$-u'$
X
v
$-u$
C
$C{:}-y$
$c{:}x$
A
$\{x - u\}_{\overline{X_C}}$
$\{x - u - u'\}_{\overline{X_C}}$
$\{0\}_{X_C}$
$\{-v\}_{X_C}$
$\{x - v\}_{X_C}$
$\{v'\}_{X_C}$
O_C

(a) Sample STNUO having labeled values generated by nLC and nUC.

(b) Sample STNUO having labeled values generated by ORC.

Fig. 3. Approach for managing values produced by two complementary rules.

Rules nLC and nUC are alternatives to rule ORC. Moreover, their application to a specific timepoint may change during the propagation of constraints. Indeed, it may be that a timepoint X becomes "strictly" scheduled w.r.t. a contingent link (A, x, y, C) during the constraint propagation. In this case, all the values determined by rules nLC and nUC and propagated by the other rules considering X and the contingent link (A, x, y, C) must be removed, and the values determined by ORC rule must be propagated. Thus, backtracking is required.

5 An Alternative Approach to Avoid Backtracking

Backtracking can slow down the actual computation of checking since it requires considering backtracking for different nodes in different phases of the constraint propagation.

Accepting the cost of more memory, it seems straightforward to properly label some derived values (the ones derived by nLC/nUC and ORC when the involved contingent link has an oracle) and use them only while they are valid.

In more detail, let us denote by $\{v\}_{\overline{X_C}}$ a value derived by using the nLC or nUC rules involving a timepoint X and a contingent link (A, x, y, C). Figure 3a shows an example of an STNUO where values determined by nLC and nUC rules and then propagated to Y are labeled.

Then, let us denote by $\{v\}_{X_C}$ a value derived by the rule ORC considering a node X and a contingent link (A, x, y, C). Figure 3b shows an example of an STNUO where values determined by ORC and then propagated are labeled.

It is also possible to have constraints that require combining different labeled values such as $X \xrightarrow{\{v\}_{\overline{X_C}}} A \xrightarrow{\{u\}_{\overline{Y_D}}} F$. In this case, we combine the values concatenating the labels $X \xrightarrow{\{v+u\}_{\overline{X_C}\,\overline{Y_D}}} F$. Such a concatenation occurs only when the labels do not contain opposite components. For example, the two constraint values $X \xrightarrow{\{v\}_{X_C}} A \xrightarrow{\{u\}_{\overline{X_C}}} F$ cannot be combined by the NC rule. Indeed, if $X \xrightarrow{\{v\}_{\overline{X_C}}} A$ was derived by the nUC rule on contingent link (A, x, y, C), while $A \xrightarrow{\{u\}_{X_C}} F$ was derived by the ORC rule on the same contingent link, then the two values are mutually exclusive and, therefore, cannot be combined in the constraint $X \rightarrow F$.

Now, let us consider a case in which a negative cycle is detected. A negative cycle is detected when there is a loop on a node having a negative value v. The label ℓ associated with v can be one of the following:

– ℓ is empty. It occurs when the negative cycle does not depend on any contingent link having its oracle. The network is not AC.
– ℓ has only one component, e.g., $\{v\}_{\overline{X_C}}$. The negative cycle depends on the values the nLC/nUC rule determines involving X and C. All values having label $\overline{X_C}$ must be ignored from now on. We say that $\overline{X_C}$ *is blocked*. If the label X_C was blocked in a previous phase of the constraint propagation, then it is impossible to have a schedule for X satisfying all constraints. The network is not AC.
– ℓ has two or more components; e.g., $\{v\}_{X_C\overline{Y_D}}$. In this case, label $X_C\overline{Y_D}$ is blocked. All values having a label containing any other combinations of X_C and Y_D ($X_CY_D, \overline{X_C}Y_D$, or $\overline{X_C Y_D}$) are still valid and they can be propagated. If all other labels obtained by the combinations of X_C, Y_D have been blocked in the previous propagations; then it is impossible to find a configuration for timepoints X, C, D that is agilely controllable. Hence, the network is not AC.

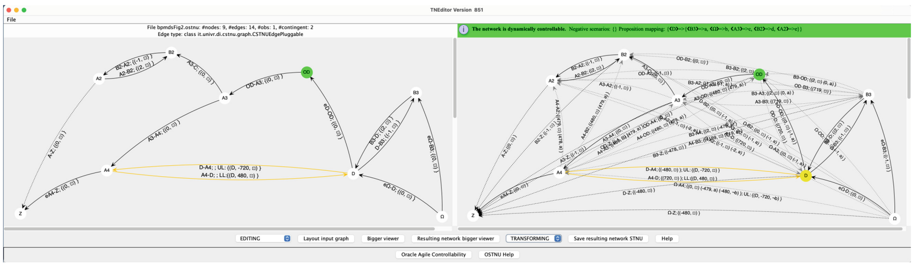
The AC-checking algorithm based on this technique made the same maximum number of rule applications, $O(n^5)$, of the MM DC-checking one because it applies a set of rules that work like MM rules. The difference in the AC-checking algorithm is that each rule application can manage more labeled values instead of just 3 like in an MM rule. A simple (gross) upper bound to the number of values that could be present on each edge is $2^{|\mathcal{T}_X||\mathcal{T}_C|}$. Therefore, the complexity is limited by $O(n^5 2^{|\mathcal{T}_X||\mathcal{T}_C|}) = O(2^{n^2})$, where n is the number of network nodes.

6 Proof of Concept

The presented approach was implemented as a proof-of-concept prototype in the (freely available) CSTNU Tool, v. 1.42 [14]. It enables users to create different kinds of temporal constraint networks and to verify automatically some properties like dynamic controllability or consistency (for some kinds of networks). In particular, it allows one to verify the agile controllability of an STNUO by the labeling technique. In case a network is agilely controllable, it returns the minimal agilely controllable network and the information of which oracles are necessary.

The screenshot from Fig. 4 shows the CSTNU Tool after the checking of the STNUO associated with the process in Fig. 1. On the left side, there is the initial network that can be edited. On the right side, there is the checked network with minimal derived constraints (edges) and the auxiliary information about the check (on the green status bar, here indicating that the process model derived from the motivating example is agilely controllable).

Currently, the tool can only manage networks having a limited number (26) of pairs of "(contingent, strictly scheduled external node)" for efficiency reasons.



File bpmdsFig2.ostnu: #nodes: 9, #edges: 14, #obs: 1, #contingent: 2
Edge type: class it.univr.di.cstnu.graph.CSTNUEdgePluggable

The network is dynamically controllable. Negative scenarios: {} Proposition mapping: {❮D❯=>{❮B3❯=>a, ❮Ω❯=>b, ❮A3❯=>c, ❮B2❯=>d, ❮A2❯=>e}}

Fig. 4. Determining Agile Controllability of an STNUO in CSTNU Tool. (Color figure online)

We conducted some experiments considering the STNU benchmarks made available in [14]. In particular, we considered 30 random instances of the sub-benchmark “DC 500”, which consists of DC STNU instances of 500 nodes (50 contingent ones) representing random temporal business processes. We reduced them to STNUO instances of 30 nodes (5 contingent and 2 oracles) and we verified that the average AC checking time results to be around 3 s. The benchmark is available at http://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.html. The Java program to view/edit/test such instances is `TNEditor` [14].

7 Discussion and Conclusion

Verifying the possibility of achieving compliance of executions with process models is typically a design-time activity [6]. Specifically for temporal processes [12,16], dynamic controllability is the most advanced property that ensures the satisfaction of all temporal requirements, irrespective of uncontrollable execution aspects [2]. Dynamic controllability has been studied in several works such as [7,8,10,17]. However, since it is not possible to take advantage of early notification of the actual duration of contingent activities for scheduling the executions, processes relying on such a possibility cannot be represented (hence verified) adequately [15].

The proposed STNU with oracles increases the modeling expressiveness for the temporal perspective of real-world processes that, until now, could not be formalized, in particular with respect to the early specification of contingent tasks. The proposed contingent durations with an associated contingent oracle resemble parameter nodes, which were introduced for the STNU in [4] and the Conditional STNU in [5]. Like contingent links, they are not under the control of the system. However, their value is revealed when the network execution starts, i.e., at time *zero*, before any other activity; additionally, parameter nodes are not associated with a duration. Conversely, an oracle node may reveal the associated contingent duration any time after the start of network execution. Therefore, existing algorithms to check the dynamic controllability of parameterized (C)STNUs are not applicable in the presence of contingent oracles.

With the introduction of Agile Controllability, we propose a new notion of temporal correctness of process models that allows verifying temporal compliance at design time for scenarios with early notification of contingent durations. A constraint propagation algorithm based on the new rules in Table 1 constitutes a possible approach to the AC-checking of an STNU with contingent oracles. A limitation of the new rule set is that it is not commutative, i.e., the result of a complete constraint propagation (until network quiescence) depends on the order of application of the rules [15]. It is possible, for instance, that the constraint propagation algorithm initially applies rules nLC and nUC, but later on, the conditions for applying such rules do not hold anymore due to the application of other rules, making the application of the initial rules overconstraining. The approach proposed in Sect. 5 based on labeling constraints makes it sufficient, during the constraint propagation, to ignore constraints with specific labels without having to delete any derived constraints that should not have been derived. While this has the benefit of avoiding backtracking and the resulting slowdown in the AC-checking, the price to pay is in terms of memory, as more complex labels have to be managed. We plan to design more sophisticated and efficient AC-checking approaches using the proposed rule set in future work.

References

1. Combi, C., Gambini, M., Migliorini, S., Posenato, R.: Representing business processes through a temporal data-centric workflow modeling language. IEEE Trans. Syst. Man Cybern. Syst. **44**(9), 1182–1203 (2014). https://doi.org/10.1109/TSMC.2014.2300055
2. Combi, C., Posenato, R.: Controllability in temporal conceptual workflow schemata. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 64–79. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_6
3. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**(1–3), 61–95 (1991)
4. Eder, J., Franceschetti, M., Köpke, J.: Controllability of business processes with temporal variables. In: ACM SAC 2019, pp. 40–47 (2019). https://doi.org/10.1145/3297280.3297286
5. Franceschetti, M., Posenato, R., Combi, C., Eder, J.: Dynamic controllability of parameterized CSTNUs. In: ACM SAC 2023, pp. 965–973 (2023). https://doi.org/10.1145/3555776.3577618
6. Hashmi, M., Governatori, G., Lam, H.P., Wynn, M.T.: Are we done with business process compliance: state of the art and challenges ahead. Knowl. Inf. Syst. **57**(1), 79–133 (2018). https://doi.org/10.1007/s10115-017-1142-1
7. Hunsberger, L.: Efficient execution of dynamically controllable simple temporal networks with uncertainty. Acta Inform. **53**, 89–147 (2016)
8. Hunsberger, L., Posenato, R.: A faster algorithm for converting simple temporal networks with uncertainty into dispatchable form. Inf. Comput. **293**, 105063 (2023). https://doi.org/10.1016/j.ic.2023.105063
9. Hunsberger, L., Posenato, R., Combi, C.: The dynamic controllability of conditional STNs with uncertainty. In: Workshop PlanEx@ICAPS-2012 (2012). http://arxiv.org/abs/1212.2005

10. Hunsberger, L., Posenato, R., Combi, C.: A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In: 22nd International Symposium on Temporal Representation and Reasoning (TIME 2015) (2015). https://doi.org/10.1109/TIME.2015.26
11. Kagedan, D., Ahmed, M., Devitt, K., Wei, A.: Enhanced recovery after pancreatic surgery: a systematic review of the evidence. HPB **17**, 11–16 (2015)
12. Lanz, A., Reichert, M., Weber, B.: Process time patterns: a formal foundation. Inf. Syst. **57**, 38–68 (2016). https://doi.org/10.1016/J.IS.2015.10.002
13. Morris, P.H., Muscettola, N.: Temporal dynamic controllability revisited. In: 20th National Conference on Artificial Intelligence (AAAI-2005) (2005). https://cdn.aaai.org/AAAI/2005/AAAI05-189.pdf
14. Posenato, R.: CSTNU tool: a Java library for checking temporal networks. SoftwareX **17**, 100905 (2022). https://doi.org/10.1016/j.softx.2021.100905
15. Posenato, R., Franceschetti, M., Combi, C., Eder, J.: Some results and challenges Extending Dynamic Controllability to Agile Controllability in Simple Temporal Networks with Uncertainties. Technical report 1/2023, Dip. Informatica-Univ. di Verona (2023). https://iris.univr.it/handle/11562/1116013
16. Posenato, R., Zerbato, F., Combi, C.: Managing decision tasks and events in time-aware business process models. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 102–118. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_7
17. Vidal, T., Fargier, H.: Handling contingency in temporal constraint networks: from consistency to controllabilities. J. Exp. Theor. Artif. Intell. **11**(1), 23–45 (1999)