Resolving Conflicts in Process Models with Temporal Constraints

Josef Lubas[1], Marco Franceschetti[1] and Johann Eder[1]

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

Abstract

Temporal constraints, temporal requirements, and temporal goals such as deadlines, durations, minimum and maximum time spans between events are important aspects of business processes. However, some of these aspects may happen to be in conflict with each other. Dynamic controllability emerged as the preferred notion expressing the absence of such conflicts. We propose a system that supports the design of processes with temporal constraints that are not in conflict. The system does not only check whether a designed process model with temporal constraints is admissible, i.e., dynamically controllable, but also delivers all the conflicting constraints that contribute to a violation of dynamic controllability. The system offers also the possibility to incrementally add and remove temporal constraints, and instantaneously check for absence of conflicts. Additionally, we provide an algorithm that computes the strictest temporal constraints between events, which can be added without introducing conflicts.

1. Introduction

Business processes typically have to observe temporal requirements or constraints such as the duration of activities, deadlines for the process execution and admissible laps between events (in particular the start and end of activities) [1, 2, 3, 4, 5]. Recently, significant progress was made to check a process model for the existence of conflicts between temporal constraints, i.e., mutually exclusive satisfaction of temporal constraints. To this end, several notions of temporal correctness, such as consistency, controllability, conditional controllability and dynamic controllability, with different degrees of expressiveness, strictness and problem complexity have been developed (see, e.g., [6, 7, 8, 9]).

The most elaborated notion for the correctness of temporally constrained business processes is dynamic controllability (DC) [10, 6, 9]. Dynamic controllability ensures that the temporal constraints are free from any potential conflict under any foreseeable circumstance [11]. Dynamic controllability is the most relaxed criterion to guarantee that it is possible to steer the execution of a business process in such a way that no temporal constraint violations occur at run-time despite uncertainties. Uncertainties arise when the process controller cannot know the precise duration of some activities (e.g., of invoked services) beforehand, and because of conditional execution paths that cannot be influenced.

Sound and complete algorithms are now available to effectively check whether specifications of temporally constrained business processes are dynamically controllable. The major approaches to check the dynamic controllability of processes are based on different kinds of

41st International Conference on Conceptual Modeling, 17–20 October, 2022, Online (Virtual)

temporal constraint networks [12], in particular *Simple Temporal Network with Uncertainty (STNU)* and *Conditional Simple Temporal Network with Uncertainty (CSTNU)*. Business process models with temporal constraints can be transformed into an equivalent STNU or CSTNU by applying a set of transformation rules [13]. So the dynamic controllability of a business process is delegated to checking whether its corresponding STNU or CSTNU is dynamically controllable. At this time, effective techniques for checking dynamic controllability of STNUs, such as the MMV- [14] and the RUL-system [15, 10] (respectively for CSTNUs [16]) have been proposed. These algorithms apply rules to infer temporal constraints until no additional constraint can be deduced (DC holds), or a contradiction is detected (DC does not hold).

Nevertheless, supporting the design of temporally constrained processes and supporting the negotiation of temporal requirements need more than checking procedures returning "true" or "false", as current procedures do. However, based on these procedures, we were able to develop a system that provides significantly greater support for designers and requirements engineers, to analyze the cause of conflicts, to incrementally add constraints, and to compute which additional constraints are feasible.

Here, we present a temporal process designer that offers the following features:

- If a process is not dynamically controllable, we provide a procedure that shows the subsets of constraints that are in conflict. A designer then might adjust some of these constraints to resolve these conflicts and achieve dynamic controllability.
- Temporal constraints may be added or deleted without the necessity to run the checking algorithm from scratch again. If a constraint is removed, all implicit constraints, which were derived from the removed constraint, are also removed and the other derived constraints remain.
- The tool is able to compute the strictest constraint (e.g., the strictest deadline, the longest or shortest possible duration of an activity, the minimum and maximum time lap between two events, etc.) without losing dynamic controllability. This is, in particular, helpful for requirements engineering where some of the constraints are fixed (e.g., laws, nature, etc.) and other temporal requirements are adjustable goals or subject of negotiations [17], which allows resolving existing conflicts.

In this paper, we base our considerations on STNUs, which are simpler and have lower complexity than CSTNUs and are therefore better suited to present the principles of the approach and the features of the temporal process designer.

The remainder of this paper is structured as follows: in Sect. 2 we introduce a process model and discuss preliminary concepts; in Sect. 3 we analyze desirable features for supporting the design of processes with temporal constraints; in Sect. 4 we present a designer tool supporting these features; in Sect. 5 we discuss related work; in Sect. 6 we draw conclusions.

2. Processes with Temporal Constraints

2.1. Process Model

In this section, we introduce a process model that covers the most commonly found control flow patterns: sequence, inclusive and disjunctive splits, and the corresponding joins. We currently consider only processes that are acyclic and block-structured, which prevents the design flaws pointed out in [18].

As regards the temporal aspect of the process model, it allows the definition of activity durations, process deadline, and upper- and lower-bound constraints between events (start and end of activities). Time in the process model is measured in *chronons*, an abstract time unit representing, e.g., hours, days, ..., with domain the set of natural numbers, thus time points can be placed on an increasing time axis starting at *zero*. Durations are defined as the distance between two time points on the time axis. For each activity, the process model specifies a duration range, i.e., an interval delimited by the minimum and maximum allowed duration for the activity execution. The process model allows the definition of contingent, non-contingent, and semi-contingent activity durations [19]. Contingent durations cannot be controlled, which means that it cannot be known in advance how long they will actually take: each actual duration is discovered at the completion of the associated activity. Non-contingent activity durations, instead, can be controlled at any time by the process controller, who can steer them to meet temporal constraints. Finally, semi-contingent activity durations can be decided by the executor, but each such duration can be chosen only up to the start time of the corresponding activity, and cannot be further changed.

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

- *N is a set of nodes n with $n.type \in \{start, activity, xor-split, xor-join, par-split, par-join, end\}$. Each $n \in N$ is associated with two events: $n.s$ and $n.e$, representing the start and end events of n.*
- *E is a set of edges $e = (n1, n2)$ defining precedence constraints.*
- *C is a set of temporal constraints:*
  - *duration constraints $d(n, n_{min}, n_{max}, dur)$ $\forall n \in N$, where $n_{min}, n_{max} \in \mathbb{N}$, $dur \in \{c, sc, nc\}$, stating that n takes some time in $[n_{min}, n_{max}]$. n can be contingent ($dur = c$), semi-contingent ($dur = sc$), non-contingent ($dur = nc$);*
  - *upper-bound constraints $ubc(a, b, \delta)$, where $a, b \in N^e$, $\delta \in \mathbb{N}$, requiring that $b \leq a + \delta$;*
  - *lower-bound constraints $lbc(a, b, \delta)$, where $a, b \in N^e$, $\delta \in \mathbb{N}$, requiring that $b \geq a + \delta$.*
- *$\Omega \in \mathbb{N}$ is the process deadline, specifying the maximum process duration.*

2.2. Temporal Semantics of the Process Model

To formally express the temporal semantics of the process model of Def. 1, we follow the approach shown in [20] and rely on the Simple Temporal Network with Uncertainty (*STNU*). The STNU is a sound formalism to encode temporal problems in the presence of contingent and non-contingent durations. Furthermore, the STNU offers sound and complete algorithms for inferring temporal knowledge and checking temporal correctness. To be self contained, we present the STNU only informally, and refer the reader to [14] for a detailed formalization.

An STNU $S(\mathcal{T}, \mathcal{L}, \mathcal{C})$ is a network composed of nodes and edges. Nodes (set $\mathcal{T}$) represent time points, and edges (set $\mathcal{L} \cup \mathcal{C}$) represent temporal constraints between time points. One special node $Z \in \mathcal{T}$ represents the origin, or *zero* time point, after which all other time points occur. Edges can be either non-contingent (set $\mathcal{C}$) or contingent (set $\mathcal{L}$). A non-contingent edge (A, B, δ) between two time points A and B represents a constraint, which the executor must satisfy, i.e., she must assign a value to B such that $B \leq A + \delta$ holds. A contingent edge, also called link, (A^C, l, u, C) between two time points A^C and C states that the value of C will be observed to be between l and u after the value of A^C.

To express the temporal semantics of a process model, the process model can be mapped into a temporally equivalent STNU, i.e., the STNU is a complete encoding of all the elements of the process temporal aspect. The mapping is realized by applying mapping rules as follows:

Definition 2 (STNU-mapping). *Let $P(N, E, C, \Omega)$ be a process model. The STNU $S(\mathscr{T}, \mathscr{L}, \mathscr{C})$ equivalent to P is obtained by applying the following rules:*

1. *The start node of the process is mapped into the STNU Z time point; the end node of the process is mapped into an end time point;*
2. *$\forall$ $n.s, n.e$, corresponding time points $n.s, n.e$ are in $\mathscr{T}$;*
3. *$\forall$ $(m, n) \in E$, with $m.e, n.s \in N^e$, a corresponding non-contingent edge $(n.s, m.e, 0)$ is in $\mathscr{C}$;*
4. *$\forall$ $d(n, l, u, type) \in C$ with $d.type = contingent$ a corresponding contingent link $(n.s, l, u, n.e)$ is in $\mathscr{L}$;*
5. *$\forall$ $d(n, l, u, type) \in C$ with $d.type = non - contingent \vee semi - contingent$ a corresponding non-contingent link for the minimum duration $lbc(n.e, n.s, -\delta)$ and a non-contingent link for the maximum duration $ubc(n.s, n.e, \delta)$ is in $\mathscr{C}$;*
6. *$\forall$ $ubc(a, b, \delta) \in C$, a non-contingent edge (a, b, δ) is in $\mathscr{C}$;*
7. *$\forall$ $lbc(a, b, \delta) \in C$, a non-contingent edge $(b, a, -\delta)$ is in $\mathscr{C}$;*
8. *Ω is mapped into a non-contingent edge (Z, end, Ω) in $\mathscr{C}$.*

The mapping rules in Def. 2 transform the start and end events of the process model into two STNU nodes (rule 1); each process node into two STNU nodes, one for the start and one for the end of the process node (rule 2). With rule 3, each precedence constraint of the process model is mapped into a non-contingent STNU edge. The two STNU nodes resulting from a process node are connected by either a contingent link having the minimum and maximum duration bounds for the node duration if the duration is contingent (rule 4), or a pair of non-contingent edges if the duration is non-contingent or semi-contingent. All temporal constraints and the process deadline are mapped into non-contingent STNU edges (rules 6-8).

With the above mapping, we obtain a formal STNU encoding that includes all the temporal elements of the process model from Def. 1.

2.3. Constraint Propagation for Checking Dynamic Controllability

As shown in [20], the mapping of a process model into an STNU allows verifying the dynamic controllability (DC) of the process model by checking the DC of its equivalent STNU. Different methods exist for checking the DC of an STNU; here, we adopt the approach based on constraint propagation, due to its ability to infer implicit temporal knowledge. Essentially, DC is checked by propagating the constraints of an STNU until either a constraint causing a negative cycle - in the sense of graph theory - is derived (then the STNU is not DC), or no more propagation is possible (then the STNU is DC). Here, we refer to constraints given by the modeler as *preexisting* or *explicit*; to constraints inferred through propagation as *derived* or *implicit*.

In this work, we adopt the RUL-system [10], which is a sound-and-complete and efficient constraint propagation system. For space reasons, we only briefly introduce the system and constraint propagation; for details we refer to [10].

The system consists of three distinct rules (RELAX, UPPER and LOWER) describing how implicit constraints can be derived from existing ones in an STNU. In general, each rule applies to three STNU nodes A, B, C connected by consecutive constraints $c1$ and $c2$ ($A \xrightarrow{c1} B \xrightarrow{c2} C$), and derives an additional constraint $c3$ ($A \xrightarrow{c3} C$). We call such a derivation a *triangular reduction*.

The constraint propagation algorithm continuously iterates over the set of all combinations of three STNU nodes, and checks for the applicability of any of the three rules, until no rule can be applied or a negative cycle is derived. A negative cycle indicates that a set of constraints are contradicting, i.e., one constraint can only be fulfilled if another one is violated.

As an example, consider an STNU fragment with nodes A, B and a constraint requiring B to happen at most 3 time units after A (edge $A \xrightarrow{3} B$). Suppose that, with the RUL-system, another constraint has been derived, requiring A to happen at least 4 time units before B. This results in the following edges: $A \xrightarrow{3} B \xrightarrow{-4} A$, hence a negative cycle, meaning that the STNU is not DC. It is easy to see that a negative cycle entails a negative self-cycle, e.g., $A \xrightarrow{-1} A$. We use this observation to speed up our implementation of the DC-checking procedure.

3. Designing Process Models with Temporal Constraints

In this section, we discuss actions that a process designer may perform when constructing a time-constrained process model, and present, for each action, how the design process may be optimized by our tool. Therefore we introduce first a conceptual model of our design approach (see Fig. 1).

The very first step for the process designer is to formalize a given process with temporal requirements. Our tool supports process modelling with an extended notation of UML activity diagrams that allows to explicitly model processes with temporal constraints (we call this extension Temporal Activity Diagrams).

In order to provide rigorous temporal support to the designer's actions, we transform the process model into an equivalent STNU, following the mapping rules described in Def. 2. This step is referenced by the *initialize()* operation in Fig. 1. An STNU is the formal representation of the temporal requirements of the process model and builds the foundation for our contribution since all operations are performed on STNU level. Once the process model has been initialized, we propose to check for pre-existing conflicts by running the *checkDC()* operation on the initial STNU. As described in Sect. 2.3, checking for dynamic controllability of an STNU is based on constraint propagation, thus requires to compute the STNU closure STNU*, i.e., the STNU with all propagated constraints. We use this fact as an advantage for an incremental design approach - only the closure STNU* is kept and updated in real time through operations of adding, removing or updating temporal constraints according to the changes in the process model.

However, the central question when designing processes with temporal constraints is whether the current process model yields a conflict (i.e., at least one constraint cannot be fulfilled without violating any other). Thus, a process designer may repeatedly check for dynamic controllability

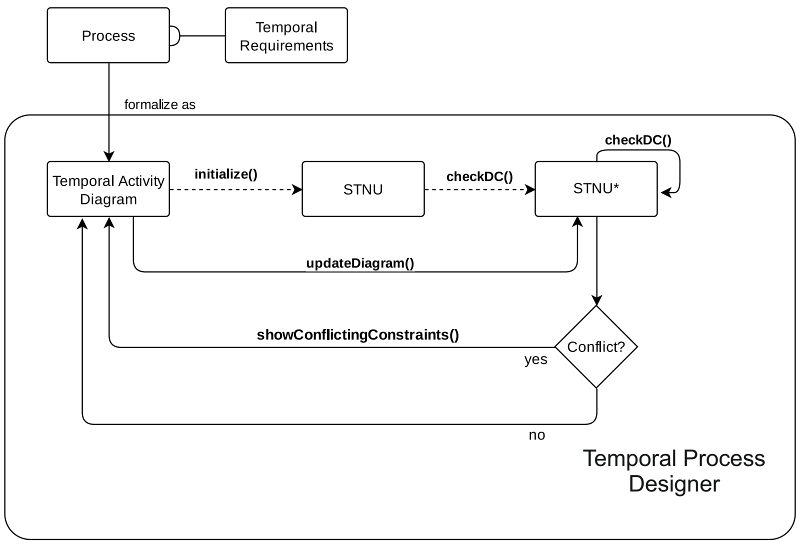

Figure 1: Conceptual model of the design process supported by the tool

throughout the design process (i.e., after updating the model). Based on whether the checking for DC yields a conflict, process design may have one out of two possible continuations:

1. If the model is DC (i.e., free of conflicts), a process modeller may go on and add further constraints. As an optimization, the complete STNU mapping is done only the first time the checking procedure is called. Further adding constraints does not require re-computing the closure with the implicit constraints. Therefore, the equivalent STNU for the DC-check is incrementally constructed by adding each time a new constraint to the STNU*. The DC-checking procedure only updates previously derived constraints if necessary.
2. If, during the design of a process, the DC-checking procedure reveals that the model is not DC (by discovering a negative cycle), with the incremental DC-checking, this means that the last added constraint yields a conflict with some preexisting constraints. At this point, a process designer needs to identify which constraints actually cause the conflict, and may check which alternative constraint would instead introduce no conflict.

In the following, we describe how we can identify conflicting temporal constraints by keeping track of the provenance of derived constraints during execution of the DC-checking procedure, and how we can compute the most restrictive constraint such that a given model remains conflict-free. In order to enhance the reader's understanding of the approach, we introduce a small running example.

3.1. Running Example

In Fig. 2 we show an example activity diagram of a process with temporal constraints. There are four activities (A, B, C, D), whereas B and C are executed in parallel. Each activity is associated

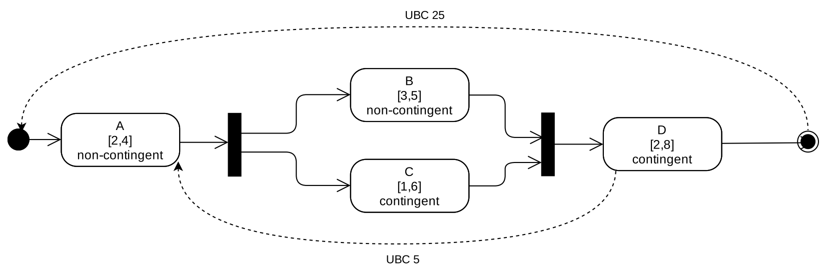

Figure 2: Running example

with a duration and duration type (contingent, non-contingent). There are also two upper-bound constraints: one requires the process to end within 25 time units, and one requires D to start at most 5 time units after A has finished.

The application of the DC-checking algorithm will derive the negative self-loop $D \xrightarrow{-1} D$, meaning that the given process model is not DC. If the system could keep track of how implicit constraints are derived, we would know that the negative self loop origins in the contradiction between the UBC that requires D to start at most 5 time units after A and the maximum duration of 6 time units of the contingent activity C. Due to the uncertain duration of C, D can start at the earliest 6 time units after A finished, resulting in the fact that it is impossible to satisfy both constraints. In the next section, we show how to identify the origin of such a conflict.

3.2. Identifying Conflicting Temporal Constraints

Given a conflict between constraints, how can a process designer determine which existing temporal constraints have to be adjusted such that dynamic controllability can be achieved? A possible solution is to keep track of how implicit constraints are derived.

For any implicit constraint c, there are exactly two other constraints c', c'' for which the application of either the RELAX, UPPER or LOWER rule derives c. So for each implicit constraint c there is a triple $p_c = \langle c', c'', r \rangle$ called *provenance* of c, where c' and c'' are constraints and r is the rule that derived c:

Definition 3 (Provenance of a Constraint). *Let $S(\mathcal{T}, \mathcal{L}, \mathcal{C})$ be an STNU. Let $c \in \mathcal{C}$ be an implicit constraint; let $r \in \{RELAX, UPPER, LOWER\}$; let $c', c'' \in \mathcal{C} \cup \mathcal{L}$ such that $c = r(c', c'')$*[1].

The provenance *of c is $p_c = \langle c', c'', r \rangle$, and it indicates that c is the implicit constraint derived from the application of rule r to constraints c', c''.*

Computing the transitive closure of the provenance of a constraint c, we can determine the preexisting constraints from which c was derived. We call the transitive closure of the provenance the *history* of c, and the preexisting nodes in the history the *origin* of c.

[1]We use $r(c', c'')$ as a shorthand to indicate that rule r is applied to c', c''.

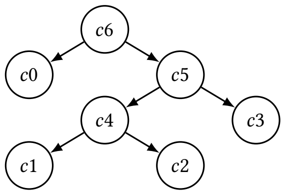

Figure 3: History of $c5$ in the form of a binary tree.

Definition 4 (History and Origin of a Constraint). *Let $S(\mathcal{T}, \mathcal{L}, \mathcal{C})$ be an STNU. Let $c \in \mathcal{C}$ be an implicit constraint with provenance $p_c = \langle c', c'', r \rangle$.*

The history *of c is the transitive closure p_c^+ of p_c.*

The origin *of c is the set of preexisting constraints $c_i^e \in p_c^+$ such that $p_{c_i^e} = \emptyset$.*

Since any derived edge by the RUL-system comes from triangular reductions, the history of each constraint can be represented as binary tree, in which internal nodes represent implicit constraints, edges from a node to its children represent the node provenance, and leaf nodes represent preexisting constraints.

As an example, let S be an STNU derived by the transformation of the example process described in Sect. 3.1, the subset of nodes $\{A.e, C.s, C.e, D.s\}$ and a subset of constraints $c0(A.e \xrightarrow{5} D.s)$, $c1(D.s \xrightarrow{0} C.e)$, $c2(C.e \xrightarrow{C:-6} C.s)$, $c3(C.s \xrightarrow{0} A.e)$. By applying the UPPER rule to $D.s \xrightarrow{0} C.e \xrightarrow{C:-6} C.s$ an additional constraint $c4(D.s \xrightarrow{-6} C.s)$ with $p_{c4} = \langle c1, c2, UPPER \rangle$ can be derived. In the next step the application of the RELAX rule on $c4$ and $c3$ derives another implicit constraint $c5(D.s \xrightarrow{-6} A.e)$ with $p_{c5} = \langle c4, c3, RELAX \rangle$. Finally, applying the RELAX rule for $c5$ and $c0$ derives the negative self loop $c6(D.s \xrightarrow{-1} D.s)$ indicating that the process model is not DC.

In order to identify which constraints cause the conflict we now compute p_{c6}^+, the transitive closure of p_{c6}, by computing the provenance of $c0$ and $c5$. Since $c0$ is a pre-existing constraint we already identified one of the conflicting constraints. For $c5$, however, we still have to compute the provenance. The origin of $c5$ are the leaf nodes $c1, c2, c3$ of the binary tree (see Fig.3). Since both $c1$ and $c3$ are precedence constraints (given by the process flow), we can determine $c2$ and $c0$ as conflicting constraints meaning that the negative self loop origins in the contradiction between the UBC that requires D to start at most 5 time units after A and the maximum duration of 6 time units of the contingent activity C. Due to the uncertain duration of C, D can start at the earliest 6 time units after A finished, resulting in the impossibility to satisfy both constraints.

To support the identification of constraints causing a contradiction with a newly added constraint, it is therefore useful to keep track of the provenance of constraints. By maintaining the provenance of each derived constraint during constraint propagation (e.g., as a binary tree), a designer can inspect the history of a constraint forming a self-loop in the case of a non-DC process model, and modify these constraints in order to resolve the conflict between constraints.

3.3. Determining the Most Restrictive Non-contingent Constraint

When process designers know the origin of a conflict between constraints, they might consider removing selected constraints to resolve the conflict. Usually, removing constraints requires re-running the DC-checking procedure, re-computing already derived implicit constraints.

An optimization is possible if we keep track of the provenance of each derived constraint generated with the modified RUL-system. We can use the provenance of each implicit constraint for optimizing the design process, by only re-computing the necessary constraints. For any removed constraint c, we remove all constraints c' where $c \in p^{+}_{c'}$. Any other constraint can be kept and does not have to be recomputed, since its history does not include c.

It may not always be possible to remove a constraint that is causing a conflict from a process model. Alternatively, a process designer may need to *adjust* existing constraints, so that the process model becomes DC, without over-constraining the model. Thus, we might ask the question: given any two nodes in an STNU, which is the most restrictive non-contingent constraint that can be added without introducing a conflict with preexisting constraints?

The most restrictive non-contingent constraint is defined as follows:

Definition 5 (Most Restrictive Non-contingent Constraint). *Let $S(\mathcal{T}, \mathcal{L}, \mathcal{C})$ be an STNU. Let $A, B \in \mathcal{T}$ be two STNU nodes. The most restrictive non-contingent constraint between A and B is $A - B \leq \delta$, with δ the minimum value such that $S(\mathcal{T}, \mathcal{L}, \mathcal{C} \cup \{A - B \leq \delta\})$ is DC, and for any $\delta' < \delta$ $S(\mathcal{T}, \mathcal{L}, \mathcal{C} \cup \{A - B \leq \delta'\})$ is not DC.*

From Def. 5 it follows that, given the most restrictive constraint $c = A - B \leq \delta$, adding any non-contingent constraint $A - B \leq \delta''$, if $\delta \leq \delta''$, will preserve DC property. However, it is sufficient to find the most restrictive constraint, since it is always possible to relax a constraint without introducing conflicts. We consider only non-contingent constraints, because contingent constraints come from external sources and cannot be controlled by the modeler.

Our proposed binary search approach to finding the most restrictive non-contingent constraint is shown in Alg. 1. The approach works as follows: for any two nodes A, B in a STNU S, we can determine the most restrictive *non-contingent* constraint $c = A - B \leq \delta$ by adding c to S, run execute the DC-checking procedure and then either increase the value of δ if S is not DC, or decrease δ if S is DC. This is done recursively, as it is a common practise for binary search. The procedure halts if S is DC and δ cannot decrease by the smallest possible value without letting S be not DC. Since we keep the provenance of derived constraints, we do not have to start from scratch every time we increase (or decrease) δ and check if S is DC: we can simply drop all constraints derived from c, and keep any other.

In case of our running example (described in Sect. 3.1), we already know that $c2$ and $c0$ are in a conflict (see Sect.3.2). We assume that $c2$ cannot be adjusted since it describes the maximum duration of a contingent activity. Thus, a process designer might wish to adjust $c0$ such that the process model is DC. The described approach will determine that for $c0(A.e \xrightarrow{\delta} D.s)$ with any $\delta \geq 6$ the process model is DC.

4. Implementation and Evaluation

We implemented the features presented in Sect. 3 into a process designer tool, and performed a series of experiments on a set of processes to show that our approach is feasible.

Algorithm 1 Compute most restrictive non-contingent constraint

```
Require: Global variables for STNU $\mathcal{S}(\mathcal{T}, \mathcal{L}, \mathcal{C})$ with $A, B, \in \mathcal{T}$ and $\Omega$ being the process maximum duration.
    Computes the smallest possible value for $\delta$, such that $\mathcal{S} \cup (A, B, \delta) \in \mathcal{C}$ is dynamically controllable.

    return $\delta \leftarrow$ B-SEARCH$(-\Omega, \Omega)$

    function B-SEARCH($left, right$)
    $mid \leftarrow (left + right)/2$
    $\mathcal{S} \leftarrow \mathcal{S} \cup (A, B, mid)$
    if DC($\mathcal{S}$) then
        $mid' \leftarrow mid - 1$
        $\mathcal{S} \leftarrow \mathcal{S} \cup (A, B, mid')$
        if DC($\mathcal{S}$) then
            return B-SEARCH($left, mid$)
        else
            return $mid$
        end if
    else
        return B-SEARCH($mid, right$)
    end if
```

4.1. A Designer Tool for Processes with Temporal Constraints

For our implementation we focused on providing an intuitive and lightweight modelling environment for processes with temporal constraints[2], rather than a fully fledged BPMN editor. Thus, we chose to use UML activity diagrams as an underlying modelling language and extended it with temporal constructs such as activity durations and upper- and lower-bound constraints according to the process model (see Def. 1).

The application was developed using ReactJs, a lightweight JavaScript library for implementing web applications. This library was chosen, because it is an easy to use and modern library with a vast amount of packages for different use cases. For drawing the graphs the library react-flow was used, because it is an highly customizeable and flexible react-library for creating graphs of all sorts. The react application also communicates with a spring webservice, which provides the main functionality of the designer tool - checking for inconsistencies and computing most restrictive non-contingent constraints.

Modelling processes is easy. A user can pick a modelling element (start/end event, activity, fork/join) an drag it to the canvas. Start/end events and for/join nodes do not have any special properties. Activities, however, require a minimum and a maximum duration and a type. Allowed types are non-contingent, contingent and semi-contingent. Inserting flow edges requires the user to connect center handles of two elements. Connecting top or bottom handles

[2]The GitLab repository for the designer tool is available under https://git-isys.aau.at/ics/Papers/temporal-process-designer

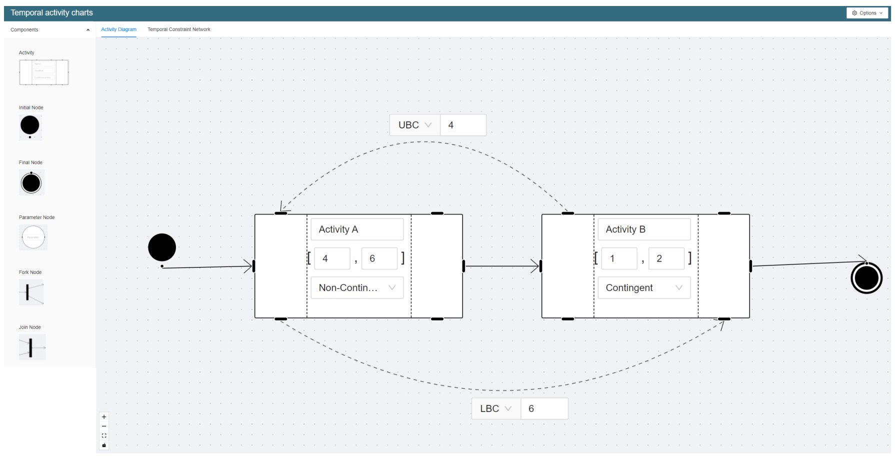

Figure 4: Overview of the designer tool

allows to insert a temporal constraint. Temporal constraints can either be UBC or LBC. As a modeler designs a process with the operations discussed in Sect. 3, the equivalent STNU is automatically constructed for the DC-check of the process model. Figure 4 shows a screenshot of the tools user interface.

4.2. Experiments and Results

In order to show the practical applicability of the designer tool, we measured the computation times for the two main functionalities 1) incremental dynamic controllability checking and 2) the computation of most restrictive non-contingent constraints on a regular Windows 10 physical machine with an i7 CPU and 16GB of RAM. As a basis, we used a set of 50 pre-generated random process models with different size and configuration of constraints. For each experiment we structured our set of process models into subsets of similar size. This resulted in 5 subsets with 10, 20, 30, 40 and 50 activities (each subset including 10 processes). With growing size, also the amount of temporal constraints within the process models increases, giving us the smallest process with a total of 25 STNU nodes and 2 temporal constraints and the largest process with a total of 129 STNU nodes and 10 temporal constraints.

In Table 1 we show the measured minimum, maximum, and average times for checking dynamic controllability on the processes of the test set.

Table 2 shows the measured minimum, maximum, and average times for computing the most restricitve non-contingent constraint with binary search (described in Alg. 1).

Total Nodes	Constraints	Min time (ms)	Max time (ms)	Average time (ms)
10	2	4	27	9
20	4	17	51	29
30	6	55	152	92
40	8	171	346	250
50	10	279	743	522

Table 1
Overview of computation times for executing the DC-checking procedure.

Total Nodes	Constraints	Min time (ms)	Max time (ms)	Average time (ms)
10	2	1	18	6
20	4	16	56	35
30	6	30	131	80
40	8	84	346	236
50	10	176	858	503

Table 2
Overview of computation times for the execution of Alg. 1.

5. Related Work

The design of business processes with temporal constraints has been considered in several prior works. In [21] the concepts of upper- and lower-bound constraints were introduced. The work in [22] addressed the representation of such temporal constraints and activity durations in a BPMN-like language. In [7] the authors raised the need to distinguish between contingent and non-contingent durations in business process, while the authors of [19] introduced the notion of semi-contingent durations. Finally, [5] posed an essential basis for the consolidation of patterns of temporal constraints in the design of business processes.

Checking the controllability of dynamic systems such as business processes is subject to extensive research, with most approaches based on mapping to some kinds of Temporal Constraint Networks ([12, 23]), or Timed Automata [24, 25].

A recent publication [26] proposes a tool for modelling business processes as time-aware extension of a the BPMN notation (TimeAwareBPMN-js). The authors adapted the bpmn-js toolkit offered by Camunda to allow modelling processes with temporal constraints on a subset of BPMN elements and provides functionality for checking for dynamic controllability. In contrast to [26], our proposed modeling tool incrementally constructs the STNU equivalent to the process model, and is able to show to the process designer the provenance of any implicit temporal constraint, aiding the adjustment of temporal constraints.

To the best of our knowledge, there exists only one technique dedicated to the problem of finding the most restrictive constraint between two time points, found in [23]. The approach in [23] introduces piece-wise linear functions to the formalism of STNUs and uses a modified version of the MMV constraint propagation system [14]. Although this approach is elegant, we propose a binary search as a competitive and in fact more simple alternative approach.

6. Conclusion

Adequate engineering of temporal requirements is essential for successful execution of time-constrained business processes. However, practitioners could not benefit from advancements in the research so far, as existing algorithms were focusing on checking given process models rather than supporting the development of correct process models. The ambition of this work was to leverage on recent theoretical results to develop a system supporting the needs and tasks of process designers to reach models free from conflicting constraints.

We developed a tool to support both designing time-constrained process models, as well as negotiating temporal requirements by assuring that the temporal constraints are not conflicting. Our approach assumes that it is within the capacity of a process controller to steer the execution of a process in a way that no temporal constraint is violated. The tool offers a number of features for verifying temporal properties (e.g., dynamic controllability, provenance of inconsistencies), and for giving guidance (e.g., which are the admissible temporal constraints) in the design of a time-constrained process.

We regard our work as a significant contribution to bridging the gap between theoretical research achievements and practical applications. For the future, we plan to extend our tool to offer more expressive temporal constraint networks, to cater for different process model notations, in particular, BPMN and EPC, and advanced temporal control structures.

References

[1] C. Bettini, X. S. Wang, S. Jajodia, Temporal reasoning in workflow systems, Distributed and Parallel Databases 11 (2002) 269–306.

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

[3] J. Eder, M. Franceschetti, Time and business process management: Problems, achievements, challenges (invited talk), in: 27th Int. Symposium on Temporal Representation and Reasoning (TIME 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[4] J. Eder, E. Panagos, M. Rabinovich, Workflow time management revisited, in: Seminal contributions to information systems engineering, Springer, 2013, pp. 207–213.

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

[6] C. Combi, L. Hunsberger, R. Posenato, An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited, in: Agents and Artificial Intelligence, Springer, 2014.

[7] C. Combi, R. Posenato, Towards temporal controllabilities for workflow schemata, in: 2010 17th International Symposium on Temporal Representation and Reasoning, IEEE, 2010, pp. 129–136.

[8] J. Eder, M. Franceschetti, J. Lubas, Conditional schedules for processes with temporal constraints, SN Computer Science 1 (2020) 1–18.

[9] L. Hunsberger, R. Posenato, C. Combi, The dynamic controllability of conditional stns with uncertainty, arXiv preprint arXiv:1212.2005 (2012).

[10] M. Cairo, R. Rizzi, Dynamic controllability of simple temporal networks with uncertainty: Simple rules and fast real-time execution, Theor. Comput. Sci. 797 (2019) 2–16.

[11] J. Eder, M. Franceschetti, J. Lubas, Time and processes: towards engineering temporal requirements, in: Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021), 2021, pp. 9–16.

[12] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial intelligence 49 (1991) 61–95.

[13] M. Franceschetti, J. Eder, Computing ranges for temporal parameters of composed web services, in: Proceedings of the 21st Int. Conf. on Information Integration and Web-based Applications & Services, ACM, 2019, pp. 537–545.

[14] P. H. Morris, N. Muscettola, Temporal dynamic controllability revisited, in: Aaai, 2005, pp. 1193–1198.

[15] M. Cairo, R. Rizzi, Dynamic Controllability Made Simple, in: 24th International Symposium on Temporal Representation and Reasoning (TIME 2017), volume 90, Dagstuhl, Germany, 2017, pp. 8:1–8:16.

[16] L. Hunsberger, R. Posenato, Sound-and-complete algorithms for checking the dynamic controllability of conditional simple temporal networks with uncertainty, in: TIME 2018, volume 120, 2018, pp. 14:1–14:17.

[17] M. Franceschetti, J. Eder, Negotiating temporal commitments in cross-organizational business processes, in: 27th Int. Symposium on Temporal Representation and Reasoning, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[18] C. Combi, M. Gambini, Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2009, pp. 42–59.

[19] M. Franceschetti, J. Eder, Semi-contingent task durations: Characterization and controllability, in: International Conference on Advanced Information Systems Engineering, Springer, 2021, pp. 246–261.

[20] J. Eder, M. Franceschetti, J. Köpke, Controllability of business processes with temporal variables, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 40–47.

[21] J. Eder, E. Panagos, M. Rabinovich, Time constraints in workflow systems, in: Advanced information systems engineering, Springer, 1999, pp. 286–300.

[22] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Toward a time-centric modeling of business processes in bpmn 2.0, in: Proceedings of Int. Conf. on Information Integration and Web-based Applications & Services, ACM, 2013, p. 154.

[23] L. Hunsberger, R. Posenato, Propagating piecewise-linear weights in temporal networks, in: Proceedings of the Int. Conf. on Automated Planning and Scheduling, volume 29, 2019, pp. 223–231.

[24] M. Zavatteri, L. Viganò, Conditional simple temporal networks with uncertainty and decisions, Theoretical Computer Science 797 (2019) 77–101.

[25] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Dynamic controllability via timed game automata, Acta Informatica 53 (2016) 681–722.

[26] M. Ocampo-Pineda, R. Posenato, F. Zerbato, Timeawarebpmn-js: An editor and temporal verification tool for time-aware bpmn processes, SoftwareX 17 (2022) 100939.