Controlled violation of temporal process constraints – Models, algorithms and results
Akhil Kumar, Russell R. Barton
Smeal College of Business, Penn State University, University Park, PA 16802, USA

A R T I C L E   I N F O
Article history:
Received 13 November 2015
Received in revised form
23 May 2016
Accepted 7 June 2016

Keywords:
Process management
Temporal workflow model
Constraint violation

A B S T R A C T
While there has been much work on modeling and analysis of temporal constraints in workflows in the context of many real-world applications, there has not been much work on managing violations of temporal constraints. In real-time workflows, such as in medical processes and emergency situations, and also in logistics, finance and in other business processes with deadlines some violations are unavoidable. Here we introduce the notion of controlled violations as the ability to monitor a running process and develop an approach based on constraint satisfaction to determine the best schedule for its completion in a way so as to minimize the total penalty from the violations. The violations are evaluated in terms of metrics like number of violations, delay in process completion, and penalty of weighted violations. We also relate our work to the concept of controllability in literature and show how it can be checked using our method. Finally, we analyze the properties of our approach and also offer a proposal for implementation.

© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

Many real-world workflows run under time constraints. A mortgage application received by a bank from a customer for the purchase of a house must go through steps like credit check, property appraisal, title search, etc. within a fixed amount of time with clear deadlines for each stage. Similarly, a business order for a desktop computer must be assembled, packed, loaded and shipped according to a clear schedule. In a hospital setting a patient proceeds through steps like testing, diagnosis, pre-operation care, surgery, post-operation care, recovery and discharge as per prescribed guidelines. These are all examples of time-sensitive service processes where timeliness has a huge impact on service quality.

Medical processes are particularly sensitive to the observance of strict temporal guidelines for the success of a treatment procedure. Some examples of such guidelines that arise in a medical process (say for the treatment of a fracture) are:

- A radiologist’s report must be submitted within 24 h of a CT scan.
- If surgery is needed it must take place within a week of the radiologist's report.
- Antibiotics must be taken for 3 days before surgery.
- A blood thinner like Aspirin must be stopped 24 h before surgery.
- The patient must recover in the hospital for 2 days before being discharged.
- The total time from patient admission to discharge should not exceed 7 days.

In modeling such time-aware processes [6,7,17,20,26], the duration of each activity (or task) is provided as a
range, or just a lower or upper limit. For example in a medical process the duration of the patient admission activity is, say, between 10 and 20 min. By associating such durations with each activity one can determine expected minimum and maximum times for each execution path of the workflow from start to end. Moreover, deviations from the expected times can be monitored, and appropriate messages and alerts can be generated to draw attention. Another aspect of temporal workflows relates to inter-activity constraints that impose restrictions on the elapsed time between one activity and another. Further they may be specified with reference to the start or finish time of the respective activities. A variety of temporal constraints can be imposed on a workflow [18]. While general types of semantic constraints have been studied in literature [15,23,24], there is less work on temporal constraints.

A temporal workflow should represent various temporal patterns and relationships among activities. Temporal patterns and ways of reasoning with them are discussed in [1,5,12]. To some extent, planning a temporal workflow is like scheduling with concepts like early (late) start times and finish times for various activities [10,11]. Another concept in the context of temporal workflows is the idea of controllability [7,17,22] which relates to the flexibility present in a workflow schedule. The work on controllability is based on the notion of conditional simple temporal networks [31] which were developed in the context of planning. A workflow that allows activity durations to fall anywhere within their allowed range and still complete successfully is said to be dynamically controllable. Algorithms for dynamic controllability are discussed in [7,17,22].

In this paper, we take the view that while on the one hand guidelines are very important, yet on the other it is not always possible to enforce them very strictly in practice. Hence, there must also be some leeway or allowance for deviations from the guidelines. Some unexpected delays may occur for various reasons at run time (e.g. patient admissions is backed up; CT machine has broken down, etc.) and lead to violations of constraints. If a task deviates slightly from its prescribed temporal range, it does not mean that the workflow cannot proceed. The natural question to pose then is: how will this deviation or violation affect the rest of the workflow? If the effect is small then the workflow can continue normally. Our goal in this paper is to develop a model that can take into account the possibility of violation of various constraints and explore the tradeoffs among the violations. Thus, if antibiotics medication has to be taken for three days before surgery and this will delay the surgery, there is a tradeoff between reducing the duration of the medication and delaying the surgery.

The novel aspect of our work is that we allow for constraints to be violated by introducing relaxation variables in our model, thus allowing for “graceful degradation.” Our approach is based on constraint satisfaction with respect to an objective function. Each temporal constraint (both intra-activity and inter-activity) can be expressed as a linear equation(s). By checking if the constraints are consistent one can verify if they will all be satisfied. These variables assume values equal to the amount of violation in a constraint to force satisfaction. At the same time we also associate penalties with each violation, e.g. for every time unit of delay in start of surgery beyond the guidelines. Finally, these penalties are aggregated and minimized in an objective function.

This paper is a comprehensive extension of an earlier work [16] and includes detailed algorithms to describe our methodology, and extensive design- and run time analyses of temporal workflows. There is an expanded coverage of repetition structures like loops, formal results, claims of correctness and completeness related to our methodology, and a proposal for implementation.

This paper is organized as follows. In Section 2 we discuss a basic model for describing temporal constraints and show how it can be translated into structural and temporal constraint equations. Then, in Section 3, we describe how the approach was implemented and tested. Next, Section 4 extends our approach for managing violations of constraints and develops a formal optimization model based on penalties. Section 5 discusses how our approach can be extended to more complex control flow structures involving overlapping and repetitive activities. Later, Section 6 discusses some analytical results that highlight the features of our approach, while Section 7 gives an implementation proposal for our methodology. Finally, Section 8 discusses related work, and the last section gives the conclusions and shares some thoughts for future work.

2. Basic notation and modeling approach

2.1. A simple temporal model

To create a temporal model of a process two types of constraint models are combined: (1) basic structural constraint model, and (2) temporal constraint model. The structural constraints capture the control flow of the process to coordinate the proper sequence in which the tasks occur. The temporal flow model considers the permitted durations of each activity and the minimum or maximum gaps between them.

Def. 1. A general temporal process model TP can be represented as:

\[ TP = (T, A, X; E, TD, TT) \]

Where

- \( T \): set of task nodes, T1, T2, ...
- \( A \): set of AND control nodes, A1, A2, ...
- \( X \): set of XOR control nodes, X1, X2, ...
- \( E \): set of edges among the nodes in \( \{ T, A, X \} \)
- \( TD \): set of task duration ranges: \( \{(Ti, Dimin, Dimax)\}_i \)\]
- \( TT \): set of additional inter-task constraints: \( \{(Ti, Tj, SiF, SiF, Ti_min, Ti_max)\}_i \)\]

Fig. 1 shows an example of a simple temporal model. It shows the control flow, along with [min,max] durations of each task and inter-task constraints. It can be expressed as:

- \( T: \{ T1, T2, ..., T6 \} \)
- \( A: \{ A1, A2 \} \)
- \( X: \{ X1, X2, X3, X4 \} \)
\[E: \{(\text{start, } X_1), (X_1, T_1), (X_1, T_2), (T_1, A_1), (A_1, T_3), (A_1, T_4), (T_3, A_2), (T_4, A_2)\}\]

\[TD: \{(T_1, D_{1\text{min}}, D_{1\text{max}}), (T_2, D_{2\text{min}}, D_{2\text{max}}), (T_3, D_{3\text{min}}, D_{3\text{max}}), \ldots\}\]

\[TI: \{(T_1, T_5, S, S, T_{I1\text{min}}, T_{I1\text{max}}), (T_4, T_5, S, F, T_{I2\text{min}}, T_{I2\text{max}})\}\]

Note that while we only consider durations of, and delays between pairs of, activities, fixed time activities can also be modeled by setting their relative time with respect to the start of a process and converting them into delays with respect to the start activity.

2.2. A constraint satisfaction approach

Next we show how to map the above model into a series of constraint equations that can be solved using a constraint satisfaction approach. We need to consider two types of constraints: structural flow constraints and temporal constraints. The flow constraints capture the coordination sequence among tasks, while the temporal constraints specify the durations for a task and also the inter-task gaps or delays.

2.2.1. Structural constraints (SC)

Structural constraints are represented by structural equations to capture the flow of a process. Each node in a process is represented by a binary 0-1 variable.

**Structural constraint representation.** SC constraints are represented by structural balance equations of a process. In doing so, each task node as well as the special tasks, start and end, are treated as variables. Thus, task ‘Start’, ‘End’, T1, T2, etc. are also variable names. These variables are all binary integers with a value 1 to denote that the corresponding task is present in an execution path or an instance of the process, and 0 to denote it is absent.

**Def. 2.** Structural balance equations. These equations represent the structural relationships among a set of binary task variables that correspond to actual tasks in a process and describe the structural behavior of a process model.

For example, the equations for sequence, choice and parallel relationships among tasks are shown in Fig. 2. Additional structural patterns are described in Table 1.

**Def. 3.** A complete structural process model is one that includes:

- one equation that captures the link of each task Ti (or connector Xi, Ai) to its preceding task(s) and/or connector(s) unless Ti is the first task in the process; and

<table>
<thead>
<tr>
<th>Structure</th>
<th>Representation</th>
<th>Constraint equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td><img src="sequence_diagram.png" alt="Sequence Diagram" /></td>
<td>(T_2 = T_1;)</td>
</tr>
<tr>
<td>Choice-Split</td>
<td><img src="choice-split_diagram.png" alt="Choice-Split Diagram" /></td>
<td>(X_1 \leq T_2 + T_3;) (X_1 \geq T_2; X_1 \geq T_3;)</td>
</tr>
<tr>
<td>Choice-Join</td>
<td><img src="choice-join_diagram.png" alt="Choice-Join Diagram" /></td>
<td>(X_2 \leq T_1 + T_2;) (X_2 \geq T_1; X_2 \geq T_2;)</td>
</tr>
<tr>
<td>Parallel-Split</td>
<td><img src="parallel-split_diagram.png" alt="Parallel-Split Diagram" /></td>
<td>(A_1 \geq T_2 + T_3 - 1;) (A_1 \leq T_2; A_1 \leq T_3;)</td>
</tr>
<tr>
<td>Parallel-Join</td>
<td><img src="parallel-join_diagram.png" alt="Parallel-Join Diagram" /></td>
<td>(A_2 \geq T_1 + T_2 - 1;) (A_2 \leq T_1; A_2 \leq T_2;)</td>
</tr>
</tbody>
</table>

**Fig. 2.** Structural balance equations for process modeling structures.
one equation that captures the link of each task $T_i$ (or connector $X_i$, $A_i$) to its succeeding task(s) and/or connector(s) unless $T_i$ is the last task in the process

**Def. 4.** A solution of a structural process model is of the form:

$$\forall T_i \in T, T_i = 0 \text{ or } 1,$$

$$\forall X_i \in X, X_i = 0 \text{ or } 1, \text{ and}$$

$$\forall A_i \in A, A_i = 0 \text{ or } 1$$

Such a solution identifies the tasks that are included in an instance (i.e. the corresponding $T_i = 1$) and the associated $X$ and $A$ (i.e. the corresponding $X_i$ or $A_i = 1$) connectors.

**Def. 5.** A correct solution of a complete structural process model is one that consists of a valid executable instance of the process model, where each connector that is part of the solution behaves according to its corresponding expression in Fig. 2.

**Def. 6.** A process model is sound iff: (1) every task and connector lies on a path from the start to the end node; and (2) each $T_i$, $X_i$ and $A_i$ appears in at least one correct solution.

**Claim 1.** A correct solution (if one exists) of a sound process model is obtained by solving the system of simultaneous (structural balance) equations in terms of the $T_i$, $X_i$ and $A_i$ variables, assuming that the start or the first task in the process is activated (or assigned a value $T_i = 1$).

Proof. The proof rests on arguing that:

1. the full set of simultaneous equations correctly represents the behavior of the model in terms of task and connector variables, i.e. $T_i$‘s, $X_i$‘s and $A_i$‘s.
2. Solving the equation system produces a correct solution, i.e. a valid process instance.

To check (1), we can examine the five structures in Fig. 2 and express them as if-then-else rules. Thus,

- For a Sequence structure, if $T_1 = 1$ then $T_2 = 1$
- For a Choice-Split structure, if $T_1 = 1$, then $X_1 = 1$; and $T_2 = 1$ or $T_3 = 1$ (but not both)
- For a Choice-Join structure, if $T_1 = 1$ or $T_2 = 1$, then $X_2 = 1$; and $T_3 = 1$
- For a Parallel-Split structure, if $T_1 = 1$, then $A_1 = 1$; and $T_2 = 1$ and $T_3 = 1$
- For a Parallel-Join structure, if $T_1 = 1$ and $T_2 = 1$, then $A_2 = 1$ and $T_3 = 1$

These rules represent the correct behavior of the various structures.

To argue (2), we note that these rules describe the behavior of each primitive structure in a process. They are mapped into an equation system. Since we assume that the first task in the process is assigned a value of 1, it will assign a value to a connector and/or task of one of the five primitive structures noted above that it belongs to. Further, since we assume that the process is sound, it means that the primitive structures are connected to each other such that every structure that is activated will produce an output, and by induction a final output is produced. Based on soundness we can argue that finally the activated tasks and connectors will represent a correct process instance. Hence, a solution to the equation system will represent a correct or valid process instance.

2.2.2. Basic temporal constraints (TC)

The temporal constraints express a variety of temporal relationships. Here we consider three types of constraints: flow constraints, task duration constraints and the inter-task gap constraints. We will discuss each one separately.

**Def. 7.** Temporal Flow (TF) constraints: These constraints are derived from the edge set $E$. For every node $n_i$ and successive node pair $(n_i, n_j)$ in $E$, we add a constraint as:

$$\text{TF}_i \leq \text{TF}_j$$

where

$$\text{TF}_i: \text{start time of node } i \text{ relative to the start time of the workflow instance}$$

$$\text{TF}_j: \text{finish time of node } i \text{ relative to the start time of the workflow instance}$$

**Def. 8.** Task duration (TD) constraints: These constraints ensure that the duration of an activity $i$ lies between the permitted range $[\text{Di}_{\min}, \text{Di}_{\max}]$. They are specified as:

$$\text{Di}_{\min} \leq \text{TF}_i - \text{TF}_i \leq \text{Di}_{\max}$$

**Def. 9.** Inter-task (TI) constraints: These constraints ensure that the gap or delay between the start (end) of an activity pair $(ij)$ lies between the permitted range $[Gij_{\min}, Gij_{\max}]$. They are specified as:

$$Gij_{\min} \leq (Tij - TSI_{ij}) \leq Gij_{\max}$$

**Def. 10.** Duration constraints for $A$ and $X$ connectors.

1. For X connectors, the duration is $XFi - XSi = 0$
2. For A-split connectors also, $AFi - ASi = 0$
3. For A-join connectors, $AFj = Max(TFi), \forall TFi$

s.t. $(TFi, AFj) \in E$

Next, we define temporal consistency and show that our approach is correct. It is also possible to give non-zero times to the $X$ and $A$ connector durations, and it will not affect our approach.
Def. 11. A solution of a temporal process model is of the form:
\[
\forall T_i, (T_i, T_{Si}, T_{Fi}), T_i = 0 \text{ or } 1, T_{Si}, T_{Fi} \in \mathbb{R}^+ \\
\forall X_i, (X_i, X_{Si}, X_{Fi}), X_i = 0 \text{ or } 1, X_{Si}, X_{Fi} \in \mathbb{R}^+ \\
\forall A_i, (A_i, A_{Si}, A_{Fi}), A_i = 0 \text{ or } 1, A_{Si}, A_{Fi} \in \mathbb{R}^+ \\
\text{if } T_i (X_i \text{ or } A_i) = 0 \text{ then } T_{Si} (X_{Si} \text{ or } A_{Si}) \text{ and } T_{Fi} (X_{Fi} \text{ or } A_{Fi}) \text{ are not valid.}
\]

Def. 12. Temporal consistency. A temporal process model is temporally consistent if for every valid and complete execution path (from start to end) there exists a solution that satisfies the duration and inter-task constraints.

Claim 2. Solving the combined set of SC and TC equations leads to a structurally correct and temporally consistent solution for a sound temporal process model.

Proof. We have already argued structural correctness in Claim 1 above by showing that the system of equations that describe the structure of the process can be solved for a sound process. The temporal constraints are described in terms of another set of variables, i.e. the start and finish times \( T_{Si} \) and \( T_{Fi} \) of each task \( i \). Using these variables we can create another system of equations to represent the temporal flow requirements. There are broadly two types of such constraints.

1. The flow constraints such as: \( T_{Si} \leq T_{Fi} \) and \( T_{Fi} \leq T_{Sj} \) (by Def. 7) represent the control flow in temporal terms as a partial order on \( TS \) and \( TF \) events since both the control and temporal flow must be satisfied.

2. The duration constraints such as: \( D_{i_{\min}} \leq T_{Fi} \) (by Def. 8) for each task (and similarly for each connector by Def. 10), along with the intertask constraints such as: \( G_{ij_{\min}} (TS_{j}) \)\( - T_{Fi} (T_{Si}) \leq G_{ij_{\max}} \) (by Def. 9) impose further temporal restrictions on the constraints.

Hence, a solution to the SC and TC equation system will satisfy both the control flow and temporal requirement in a consistent way.

It should be stressed that the above formulation can lead to multiple solutions since the durations of all activities fall within a range. To find a single solution, we should set an objective such as to minimize the total execution time of the process instance. In the next section we will explain our solution method. Further, note that this is a kind of weak consistency in which it shows that a schedule exists for the temporal workflow for one combination of durations for each activity. It does not show that all combinations of durations in the allowed range will lead to a valid solution as in dynamic controllability \([14]\). Moreover, solving this system of equations will give one solution but other solutions can be found by assigning specific values to certain variables.

2.3. Additional temporal constraints

Table 2 gives a summary of various kinds of temporal constraints, categorized in three groups: basic, overlap and repetition constraints. The constraints 1 and 2 are the basic ones discussed above. Constraints 3-5 are overlap constraints. Finally, constraints 6-9 are repetition constraints. This shows that we can also represent more complex constraints involving combined durations of activities and overlap among activities, as well as express loops and restrict the number of times a loop is repeated. All of these constructs have practical applications which will be discussed later. We will also argue completeness of our approach later too.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Meaning</th>
<th>Formal Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>1. Duration of a task ( T_i )</td>
<td>Duration of a task ( T_i ) is between ( t_{\min} ) and ( t_{\max} )</td>
</tr>
<tr>
<td></td>
<td>2. Minimum (maximum) gap between ( (T_i, T_j) )</td>
<td>Specify minimum (maximum) gap between two tasks</td>
</tr>
<tr>
<td>Overlap</td>
<td>3. (CO) Combined overlap ( (T_i, T_j, T_{k}, \ldots) )</td>
<td>Duration for which all ( n ) tasks are overlapping with one another</td>
</tr>
<tr>
<td></td>
<td>4. (CD) Combined duration ( (T_i, T_j, T_{k}, \ldots) )</td>
<td>Duration from the start of the first to start, to the end of the last to finish (Time span)</td>
</tr>
<tr>
<td></td>
<td>5. Pair-wise overlap ( (T_i, T_j, T_{k}, \ldots) )</td>
<td>Duration for which at least two or more tasks overlap</td>
</tr>
<tr>
<td>Repetition</td>
<td>6. ( r )-dependency ( (T^{(k)}, T^{(k+1)}, g) )</td>
<td>( T^{(k+1)} - T^{(k)} \leq g )</td>
</tr>
<tr>
<td></td>
<td>7. Alternating (( T_i, T_j ))</td>
<td>Every occurrence ( k ) of ( (T_i, T_j) ) alternates</td>
</tr>
<tr>
<td></td>
<td>8. Maximum number of repetitions of a loop</td>
<td>Task ( T_i ) must not repeat more than ( r ) times.</td>
</tr>
<tr>
<td></td>
<td>9. Max repetitions in a time interval</td>
<td>No more than two successive occurrences must occur within a duration ( D )</td>
</tr>
</tbody>
</table>

Table 2

Summary of modeling constructs for temporal constraints.
describe a temporal process model. The two sets of constraints are combined to obtain a complete model and solved to check for consistency by a constraint satisfaction tool such as CPLEX [8]. If a solution is found it means that the model is temporally consistent; otherwise, it is not. Here we will first describe a formal approach for converting a set of constraints into a formal model. Then we will introduce a realistic example to illustrate the approach. Later we will demonstrate the approach by solving the model both at design and run times.

3.1. Formal approach for building a CPLEX model from a process model

The starting point of this work is the temporal process model. The basic idea behind our approach is to develop an optimization model in which each structural and temporal constraint is represented. Fig. 3 gives an overview of the main steps in a general approach to building such models. The structural constraints are developed by combining the individual constraints for each pattern in the process model. The constraints for each basic pattern are shown in Fig. 2. By combining the constraints for each pattern we can create a complete structural model. The structural constraints for the advanced patterns are taken directly from Table 1.

The temporal constraints are summarized in Table 2. The basic constraints are the most common ones. They can be used to specify the duration of a task and inter-task durations. Each task is assigned a start time and finish time variable, TSi and TFi respectively. By combining these variables it is possible to specify all duration and inter-task constraints.

In addition we need to specify temporal flow constraints to capture the temporal connectivity in the process. Whenever there is a direct causal link from one element of a process model (say, a task or a connector) to another this must be shown by a constraint to reflect the temporal order in which these two elements must occur. Thus, if Ti (or Xi, Ai) precedes Tj (or Xj, Aj) the temporal order is expressed as: TFi ≤ TSj. For completeness every direct causal temporal duration is captured. Formally, this constraint is expressed as: Ti = 1 && Tj = 1 → TFi ≤ TSj. This is to ensure that the constraint applies only when both the tasks are present in an actual execution instance.

Table 2 also shows how overlap and repetition constraints are represented. The detailed discussion of how to represent such constraints will be provided in Section 6. By combining the structural constraints, temporal duration, inter-task and flow constraints it is possible to have a complete set of constraints. In general such a model will have a large number of solutions as stated earlier. To restrict the number of solutions, we add an optimization function to minimize the finish time of the last task in the process. The optimization function is written as: Minimize TF[‘end’] where ‘end’ is the last task in the process model and TF[‘end’] is the finish time for this task.

This formulation is called a mixed integer linear programming (MILP) model. It can be solved in a standard tool such as CPLEX or Solver. If a solution is found it means that the problem is feasible.

3.2. A complete example to illustrate our approach

To illustrate our approach, the example in Fig. 4 of a patient suspected of having a proximal femoral fracture will be used as a running example. This figure shows a simplified clinical pathway in BPMN notation. This process model consists of 14 tasks coordinated by sequential flows along with choice and parallel structures in BPMN notation. Briefly, after a patient is admitted (T1), she undergoes anamnesis and examination (T2). Depending upon the result of the examination, if the patient is under suspicion of having a proximal femoral fracture, she has to take a CT scan test (T5); otherwise, she is diagnosed further and prepared for therapy (T3), followed by customized therapy A (T4). Alternatively, depending on the results of her imaging diagnosis (T6), she is either treated with therapy B (T7) or by surgery (T11). If surgery is needed, then it must be scheduled (T8), and two prerequisite tasks surgical planning (T9) and administering pain medication (T10), are carried out in parallel prior to T11. Recovery (T12) follows surgery (T11). Finally, the case is documented (T13) and the patient is discharged (T14).

Fig. 3. The full procedure for constructing an optimization model.
The duration for each task is written in square brackets adjacent to it (in time units). If a value is blank it does not apply. An inter-task constraint is represented by a dashed line connecting the pair of tasks to which it applies. In Fig. 4, there is a constraint between T1 and T2 that requires that T2 must finish no more than 30 time units after the start of T1. When the dashed line connects the left boundary of a task, it means that the constraint applies to its start time, while if it connects the right boundary of a task it applies to its finish time.

Following the general approach described above, the formulation is given in Fig. 5. It shows four groups of constraints. These constraints form a system of equations in 14 binary task variables (T1, T2, ...) for the presence or absence of the 14 tasks and 14 finish time (TF1, TF2, ...) floating point variables for the 14 tasks also. In addition there are corresponding variables for the connector nodes as well such as X1, XS1, XF1, ..., A1, AS1, AF1, .... All the TD and TI constraints are preceded by a condition check to see if the task(s) to which they apply are activated in the process path. Otherwise, the conditions would not apply.

This system of equations can be solved for a solution. However, in general this problem will have an infinite number of solutions since we are dealing with real-valued variables for generality. Hence, we added an objective function to create a mixed integer programming formulation (MILP), where the objective is to minimize the finish time of the last task ‘End’.

We used the tool CPLEX to solve the formulation. CPLEX is a well-known tool for solving such MILP models. It offers several operators such as if-then, min, max, count, etc. for representing various constraints.

3.3. Design time and run time solutions

Fig. 5 shows the (partial) formulation for the process model of Fig. 4. To create this formulation we combine the TF, TD, TI and connector duration constraints (see Def. 8-10). In addition, we add the objective function. The TD and TI constraints are written in the if-then (→) style because they apply only if the associated tasks are present in a solution.

Minimize TS['End'] //Minimize end time of process s.t.
//Structural constraints (SC)
Start. Start = 1;
End. End = 1;
SF0. T1 = Start;
SF1. T2 = T1;
SF2. X1 = T2;
SF3. T3+T5 = X1;
SF4. T4=T3; T6=T5; X2 = T6;
...
//Temporal flow constraints (TF)
TF1. TF1 ≥ TS1;
TF2. TF2 ≥ TF1;
TF3. XS1 ≥ TF2;
...
//Temporal duration constraints (TD)
TD1. T1 = 1 → TF1 – TS1 ≥ 5;
TD2. T1 = 1 → TF1 – TS1 ≤ 10;
...
//Temporal inter-task constraints (TI)
T11. T1 = 1 & & T2 = 1 → TF2 – TS1 ≤ 30;
T12. T11 = 1 & & T8 = 1 → TS11 – TS8 ≥ 100;
T13. T11 = 1 & & T8 = 1 → TS11 – TS8 ≤ 140;
T14. T11 = 1 & & T10 = 1 → TF11 – TS10 ≤ 250;
T15. T11 = 1 & & T10 = 1 → TS11 – TF10 ≥ 20;

The CPLEX solution for this MILP model produced an objective function value of 210. This is a design time solution showing that the problem is feasible. Tasks 1-4, 13-14 are present in this solution. This corresponds to the path in Fig. 4 where the upper outgoing branch is taken at X1. The solution also shows the start and finish times of each activity in this solution.

Clearly, other outgoing branches could have been taken at X1 and then at X2. To find a design time solution that shows whether these paths are feasible and the actual solution, we can force the lower outgoing path at X1 by adding a constraint T5 = 1. This produced a solution with an objective function value of 465 and included tasks 1–2, 5–6, and 8–14. Finally, we tried forcing a solution with the
upper outgoing branch at X2 by adding T7 = 1. In this case the objective function value is 255 and it includes tasks T1–T2, T5–T7, and T13–T14.

Above we developed an "ideal," design time solution that can be planned even before the process starts running. At run time changes have to be made to the formulation in a running process based on its actual progress. So, say we have just completed task T8 (schedule surgery) and while its planned completion time was 45, the actual completion time was 65 because of scheduling issues and so manual intervention was needed. We wish to find out what constraints if any will be violated on account of this delay and how the process will run from here on. Hence, we perform the steps shown in Fig. 6.

These steps are carried out in the various run time scenarios that follow.

Run time scenario 1. Consider the situation where task T8 is delayed. Its normal duration is [10, 20], but in this case it has taken 30 time units as shown below:

\[ TS_5 = 0; TS_6 = 5; TS_7 = 30; TS_8 = 35; TF_1 = 5; TF_2 = 10; TF_3 = 30; TF_6 = 35; TF_7 = 65; \]

To analyze the subsequent run time behavior of this process, we add these constraints to our formulation and remove the duration constraints for these tasks (i.e. 1, 2, 5, 6, 8) since they are already completed. Upon solving the new formulation, a new solution with a finish time for the process of 485 is found. When we set TF_8 = 75 the finish time increased to 495. However, on setting TF_8 = 85, no solution was found. Upon further inspection it was realized that constraint T13 (see Fig. 4) was being violated.

In general, if a design-time solution exists, but a run time solution is not found then it means that the problem has become infeasible on account of either duration or inter-task constraints. In this situation it is helpful to know the reason for the violation and its extent. For instance we would like to know that a certain constraint C1 is violated by X1 time units. To deal with such situations we will introduce the notion of constraint violations in the next section.

Process changes. Our approach can also deal with process changes easily. If the process changes through insertions and deletions of activities, or if activity durations are modified, only the link and activity information file needs to reflect these changes and then a new formulation can be generated easily with our program. Moreover, it is very easy to force certain branches in the process, say, at an X-split connector, by adding a constraint like Ti = 1 where Ti is the first task on the desired path.

"What-if" analysis. Further, various kinds of what-if analyses can be performed with this approach. To find the maximum possible duration of a task (say, T8) without violating any constraints, we modify the objective function to: Maximize (TS_5–TS_8["End"].), and solve the new MILP. We can also assign specific duration values to a combination of tasks and check if these values still lead to a feasible solution.

4. Modeling temporal constraint violations with relaxation

4.1. Relaxation variables

So far, we assumed that every constraint was strict and could not be violated. The main idea for dealing with violations is to introduce relaxation variables for each constraint in such a way that if a constraint is violated then the variable takes on a positive non-zero value, and otherwise it is 0. Thus a duration constraint such as, say, "schedule surgery" (T8) with a duration of [10, 20] is expressed as:

\[ TF_8 - TS_8 \geq 10 \]

By introducing a constraint variable CD_8, we may rewrite this constraint as:

\[ TF_8 - TS_8 + CD_8 \geq 10 \]

Now, CD_8 is simply a relaxation variable that assumes a non-negative value. If the actual duration of T8 is less than 10, say, the duration is 5, then CD_8 = 5. Thus the constraint is satisfied and CPLEX can find a solution for the formulation. CD_8 is an example of a lower bound relaxation variable. The upper bound constraint in our example on the duration of T8 is 20. In this case a similar relaxation variable, say CD_8', is introduced as follows:

\[ TF_8 - TS_8 - CD_8' \leq 20 \]

Notice that the negative sign before CD_8' means that the relaxation allows us to satisfy the upper bound constraint by again assuming a positive value. If the duration lies between 10 and 20, then CD_8' is 0.

Relaxation for inter-task constraints is modeled in the same way. Constraint T12 of Fig. 4, for instance, is:

\[ TS_11 - TS_8 \geq 100 \]

Again, by adding a new relaxation variable, say CI_2 we can rewrite this constraint as:

\[ TS_11 - TS_8 + CI_2 \geq 100 \]

The corresponding upper bound constraint T13 in Fig. 4 is relaxed by another relaxation variable CI_3 as follows:

\[ TS_11 - TS_8 - CI_3 \leq 140 \]

Run time scenario 2. When we revised these constraints by adding the relaxation variables in the ILP formulation, and solved it for the case where TF_8 = 85, we found a solution with an objective function value of 505...
for TF14. Moreover, the value for CI3 was 10 indicating that constraint TI3 was violated by 10 time units due to the delay in completion of T8.

### 4.2. Controlled violation with penalties

Our notion of controlling a temporal process subject to violations is that it is not sufficient to simply detect a violation and stop there. It is necessary to explore the effects of a violation that has occurred further and suggest corrective action. In the above example for run time scenario 2 we noticed that the delay in the surgery scheduling activity leads to the violation of constraint TI3 by 10 and also an overall delay of 50 in the completion of the process. Hence, it is necessary to explore further to see if there is any corrective action possible to: (1) rectify the violation in constraint TI3 by making changes to the succeeding tasks; (2) reduce the delay in completion time of the process. It is also useful here to distinguish between strict and violable constraints.

**Run time Scenario 3.** If, say, constraint TI3 is strict, then we would like to see whether there is an alternative solution that would restore TI3 but force changes in another constraint. In order to check for this, we remove the relaxation variable CI3 for the constraint TI3 to make it a strict constraint and add relaxation variables to the remaining constraints TI1, TI2, TI4, TI5. In addition, we also add the corresponding relaxation variables CI1, CI2, CI4 and CI5 to the objective function because we wish to minimize the extent of the violation. Now, we found a solution in which CD10 = 10 and TF14 = 495. This means that by reducing the duration of pain medication (T10) by 10 time units, we are able to satisfy constraint TI3 and find a solution.

### 4.3. Associating Penalties with slack variables

Above in Run time scenario 3, we assumed that the objective function was:

\[
\text{Minimize } TSI'[End'] + \sum CIi + \sum CIi
\]

This means we added the violations due to each relaxation variable taking a non-zero value to the finish time of the process. However, this objective function treats each violation equally. In real practice, it is likely that the various constraint violations might have a differential impact on the outcome. Hence, in general a different penalty may be assessed for the violation of each constraint. Such penalties would be determined by the domain experts. Therefore, the revised objective function should be:

\[
\text{Minimize } TSI'[End'] + \sum PDi * CDi + \sum Pli * Cli
\]

**Run time Scenario 4.** In our running example, a doctor may feel that violation of constraint CI5 is less important than a violation of the other constraints. Hence, we could assign a penalty of just 0.5 to CI5 and of 1 to the other constraints. When we rerun the MILP with this change we get a solution in which CI5 = 20. Now the process completion time TF[14] drops to 485. Clearly, since the penalty for CI5 is smaller the optimal solution is one which relaxes this constraint as much as possible to minimize the objective function value.

In general, each constraint may have a different penalty assigned to it based on the domain knowledge. In this way it is possible to evaluate the tradeoffs between different constraints and find different solutions at run time.

### 4.4. Controlling the extent of violation

In addition to associating weights with penalties it should also be possible to limit the extent of the violation in a duration or inter-task constraint. While some violation is reasonable, beyond a certain point it may be unacceptable. To restrict the amount of relaxation of a constraint, we could add a constraint like CI5 ≤ 10 to specify that the maximum relaxation (or violation) allowed in constraint TI5 is 10, i.e., the gap between stopping pain medication (T10) and the start of surgery must be at least 10 time units.

**Run time Scenario 5.** To illustrate this new condition, we modify the formulation of our running example by adding constraint CI5 ≤ 5 to the formulation in run time scenario 4. This means that now the constraint TI5 can only be relaxed up to 5 time units. Thus, the gap between T10 and T11 that should normally be at least 20 can just be reduced to 15 but no further. On solving the formulation now, we get a solution with CI5 = 5 and CD10 = 5. This means that in addition to reducing the gap between T10 and T11 by 5, we must also reduce the duration of the pain medication from 80 to 75 time units. The new process completion time TF[14] is now 495.

### 5. Further extensions

A key feature of our optimization based approach is that a variety of temporal patterns can be handled with this method. The various patterns were organized in Table 2 into three categories: basic, overlap and repetition. So far the focus of this paper was on the basic patterns. Now, we will discuss advanced patterns like overlap and repetition in detail.

#### 5.1. Overlap patterns

Overlap patterns allow constraints that specify a minimum or maximum amount of overlap between two or more activities. In general, for \( n \) overlapping tasks, their combined overlap period (when at least two or more tasks overlap) is given by:

\[
\text{Max}(\text{Min}(TF_i, TF_j, TF_k, \ldots) - \text{Max}(TS_i, TS_j, TS_k, \ldots), 0)
\]

To illustrate the use of this pattern, take the example in Fig. 7 that is a variant of our running example in Fig. 4. Here we have added a second pain medication (T15) in parallel with the first one (T10), but with the additional requirement that two medications (say, aspirin and morphine) may be taken together only for 20 time units.
To capture this idea, we can write a constraint such as:

\[
\min(TF_{10}, TF_{15}) - \max(TS_{10}, TS_{15}) \leq 20
\]

Appendix A shows how Min and Max functions can be linearized. Hence a constraint that uses these functions can also be linearized.

Overlap Scenario 1. We modified the MILP formulation to include this change and solved it for design time. CPLEX still gave us a best solution of 465 for the revised process. On further inspection, it was realized that the new task T15 is scheduled in such a way that TS10 gave us a best solution of 465 for the revised process. On include this change and solved it for design time. CPLEX still

\[
485. \text{Finally, to represent increase. However, if we reduce the overlap between T10 and T15 ensures that the total process time does not increase. However, if we reduce the overlap between T10 and T15 to 0, then the process completion time increases to 485. Finally, to represent containment, say task i is contained in task j, we write, Tsi \leq Tsj; and Tfi \leq Tfj.}

5.2. Repetition patterns

The general case of a loop around a task, say, Task[i], is discussed next. We assume that the maximum number of repetitions is max_RPT and the actual number is num_RPT. In general multiple instantiations of Task[i] within the loop are referred to as Task[i](k), where k is the kth instance or occurrence of the loop. Thus, the first time the task in the loop occurs is referred to as Task[i](1), the second as Task[i](2), etc. Also, note that T[i] becomes a composite task in that it encompasses multiple instances of itself. The start time of T[i] is TS[i](1), and its finish time is max(TF[i](1), TF[i](2); ... TF[i](Num_RPT)).

The actual constraints for describing a loop construct around Task[i] are shown in Fig. 8 in pseudo code similar to CPLEX syntax. Constraints 1 through 5 in the figure instantiate as many successive instances of Task[i] as the number of repetitions of the loop and assign the duration range to each one. Constraints 6 and 7 enforce the flow constraints that require the finish time of each instance to be no sooner than its start time, and the start time of each successive instance to not precede the finish time of the previous instance. Constraints 8 and 9 assign start and finish times to the composite task Task[i] based on the start time of the first instance and the finish time of the last instance. Finally, constraint 10 initiates the successor task after Task[i]. An alternative and conceptually simpler approach to modeling a loop is to “flatten” it out by having multiple repetitions of the instances of Task[i].

Repetition Scenario 1. Here we extend the Overlap 1 scenario described above and add a loop around task T5, perform CT scan. The additional constraints for a loop described in Fig. 9 are added to our formulation. We assume max_RPT = 5 and num_RPT = 3. When we run this on CPLEX with the number of repetitions set to 3 it gives a process completion time of 505. In this case since one repetition of T5 takes 20 time units, three repetitions take 60 time units. Hence, the finish time increases from 465 in the Overlap 1 scenario above to 505. Similarly, if the number of repetitions is 5, then the finish time increases to 545.

Repetition Scenario 2. Here we consider a more complex repetition scenario where a loop spans two activities T5 and T6 in our running example. The modification is shown in Fig. 9. In this example a CT scan is performed and then evaluated. If a problem was found in the scan and the report was not complete for any reason, the scan must be repeated and reevaluated. Moreover, the repetitions must be separated by an interval of 150 and there cannot be more than three repetitions (i.e. max_RPT = 3) to prevent excessive radiation to the patient.

The basic modeling principle is again the same as described above for the case of a loop around one task. However, some additional constraints must be added to the formulation to capture the fact that: (1) in each iteration of the loop T6 follows T5; (2) the start time of T5 in each successive iteration of the loop must have a minimum gap of 150 from its start time in the previous one; and (3) T5 and T6 must have the same number of iterations. These changes were made to the CPLEX formulation from the Repetition 1 scenario above and it was rerun with num_RPT=2 and 3 alternately. For two repetitions the process instance finish time was 615, and for three it was 765. This reflects the fact that each T5–T6 cycle after the first one takes 150 time units.

6. Analysis of the modeling approach

Here we first develop two formal claims that describe properties of our approach. Then we offer two remarks and discussion on the structural and expressive power of our approach.
6.1. Formal Claims

**Claim 3.** If there are no inter-task constraints, then a temporal solution can always be found by satisfying the duration constraints.

Proof. When there are no inter-task constraints, each task must only observe its duration constraints. Thus, if two tasks, say Ti and Tj, are in sequence, Tj ≥ Ti. If there is an XOR-join connector after tasks Ti and Tj leading to Tk, then Tk ≥ TFi or Tk ≥ TFi. Finally, if there is an AND-join connector after tasks Ti and Tj leading to Tk, then Tk ≥ TFi and Tk ≥ TFi. Similar constraints apply to XOR- and AND-split connectors also. Given a start time for the first task (or a successive task) as TS1 (or TSi), the start time for the following task(s) can be determined by applying these flow constraints to TF1 (or TFi). In general, these flow constraints can always be satisfied based on actual durations of each successive task.

**Def. 13.** A task duration or an inter-task duration constraint in a temporal process model is safe if solutions can be found for all values within the range of the constraint.

**Claim 4.** If a solution can be found at the two extreme values of a constraint then the constraint is safe.

Proof. By contradiction. Assume a solution cannot be found at a duration value of task i s.t. TDi = TDi_min + Δ, where 0 < Δ < Δ_max (where Δ_max = TDi_max − TDi_min), but it exists for TDi = TDi_min and TDi = TDi_max for an upper bound constraint. Any upper bound constraint between tasks i and j can in general be expressed as a linear function of task and gap durations as: TDi + TD_j + TD_k + gap_ij + gap_jk + ... ≥ U (the gap durations can be adjusted to force all connector durations to 0). Since the constraint is additive, and is satisfied at both TDi = TDi_min and TDi = TDi_max, it follows that it must be satisfied for the entire duration range of TDi_min < TDi < TDi_max. We can reason similarly for a lower bound constraint like: TDi + TD_j + TD_k + gap_ij + gap_jk + ... ≥ L.

6.2. Expressiveness of the approach (structural and temporal)

Next we present remarks and discussion of the structural and temporal expressive power of our approach.

**Remark 1.** The structural modeling approach described in this paper can represent sound processes developed from basic structures of sequence, choice-split, choice-join, parallel-split, parallel-join, m-of-n splits, m-of-n joins and any new pattern that can be described by a set of structural equations in terms of task and connector variables.

**Discussion.** Our approach is a constructive one in that it uses basic structural patterns that can be combined together to create a complete process. The behavior of each pattern is represented by means of linear equations where variables signify the presence or absence of each task. These variables can be combined in a variety of ways to create new structural constraint patterns. This gives considerable flexibility for process structure design. For example, a 2-of-3 (or in general, m-of-n) constrained OR pattern [29] may require that 2 out of 3 outgoing branches at an AND-split connector should be activated, but restrict the activation to just branches 1 and 2, or 2 and 3, and not allow 1 and 3. This pattern can be captured in the structural equations such as T1 + T2 = 2 and T2 + T3 = 2, where T1, T2, T3 are task variables corresponding to the tasks on the three different branches at the AND-split connector. This shows the structural modeling flexibility that is possible with this method.

**Remark 2.** The temporal modeling approach described in the paper can express: aggregation operations such as sum, difference, equal to, greater than, less than, min, max applied to (start, finish) temporal events of a task, and intervals derived from these temporal points by applying the above operations in a valid manner to develop constraints pertaining to these temporal events and intervals, or their combinations.

**Discussion.** Again, the temporal model allows us to express the temporal events in terms of the start and finish times of each task of the process. These temporal points can be combined to derive durations of tasks and intervals between tasks. These durations and intervals can in turn be combined using a variety of linear functions and operations to represent a large range of valid constraints. Thus, to express a constraint such as: If task T1 starts within time interval (r1, r2), then task T2 must start within time interval (r3, t4). We can express this as a rule: If T1 > r1 and T1 < r2, then T2 > r3 and T2 < t4. This rule can in turn be expressed as a set of linear constraints as shown in Appendix B. To be sure the linearization requires multiple constraints, but many solvers allow a user to...
specify a constraint as a rule and convert it into constraints automatically. Again, this illustrates the flexibility of the temporal modeling approach.

7. Implementing the approach

In this section we propose a preliminary architecture and describe how an end user can actually interact with a temporal modeling tool and also examine what-if scenarios through a spreadsheet interface that shows the results of the optimization. In the spreadsheet interface, the user can modify actual durations and see how violations are affected by doing so. Then she can also take decisions and notify the optimization module interactively.

A high level interaction architecture is shown in Fig. 10. It describes only the main components related to temporal optimization. A user will design the temporal workflow through a designer module. At run time a process manager will initiate an instance of this process in the workflow engine. The engine will pass the state of the running instance to the optimizer module that will run the model and generate one or more scenarios. These scenarios will be displayed to the user on a screen as shown in the figure. The screen will display the current status of a running workflow at the top and a spreadsheet at the bottom to illustrate possible scenarios for resuming the process execution. A user may select a scenario on this screen and click ‘Accept’ to notify the system which scenario is preferred. Alternatively a user may also generate a new scenario and click on the ‘Check’ button to have the system check this scenario. If no error is generated, the user may click the ‘Accept’ button to have this scenario accepted. In this way the user can interact with the system and even override the system scenarios. Next we discuss the use of the spreadsheet briefly.

As also noted earlier, every constraint can be written as a simple sum in terms of task durations (TD) and gaps (g). In general, an inter-task constraint between tasks i and j with intermediate tasks k and l is written as:

\[ TI_{ij} = TF[j] – TS[i] = TD[j] + g[i,k] + TD[k] + g[k,l] + TD[l] + g[l,j] + TD[j] \]

For analysis purposes we can display the constraints in a table format as shown in Table 3. A binary 0-1 value for each gap or duration entry indicates if it belongs to the constraint. The last two columns show the type of constraint (≤, ≥) and its bound. These constraints correspond to the ones discussed earlier in our running example, with one exception. T12’ is a new constraint that captures the fact that the two parallel paths between A1 and A2 must take the same amount of time, i.e. the sum of the task durations and gaps along each path must be equal.

Once the constraints have been captured in this format, then we can check the satisfaction of each constraint against actual task and gap durations by a simple operation that can be performed in Excel. This produces the analysis spreadsheet as shown in Table 4, which the end user will see (Fig. 10).

This spreadsheet displays (in row 1) one system generated scenario of task (TD) and gap (g) durations. Then, in the right part of the spreadsheet the constraint violations are calculated as the dot-product of the binary values in Table 3 and the actual durations in Table 4 using the Excel sum-product function in a formula. The formula also compares the result with the right-hand side value of the constraint to decide if there is a violation and if so by how much. Row 1 of the spreadsheet shows the scenario where task T8 is delayed by 30 time units. In this case one can see that T13 is violated by 10 time units.

Rows 2 and 3 show scenarios that are generated by a user or a doctor by adjusting the values in row 1. To satisfy T13, a user may reduce g[A2,T11] from 20 to 10 (see row 2). But now constraint T15 is violated by 10 time units. By changing the TD and/or ‘g’ values on the Excel spreadsheet, we can still not satisfy all constraints. Finally, row 3 illustrates a scenario where to limit the violation of T15 to 5 time units a user decided to reduce the duration of T10 (pain medication), i.e. TD[T10] by 5 time units from 80 to 75.

Fig. 10. An architecture for realizing our approach.

| Table 3 |
| A subset of constraints in the spreadsheet format. |
|------------|--------|---------|--------|---------|-----------|--------|---------|--------|------|-------|
| T12        | 1      | 1       | 1      | 1       | 1         |        |         |        | ≥    | 100   |
| T13        | 1      | 1       | 1      | 1       | 1         |        |         |        | ≤    | 140   |
| T12’       | 1      | 1       | 1      | 1       | 1         | –1     | –1      |        | =    | 0     |
| T14        | 1      | 1       | 1      | 1       | 1         |        |         |        | ≤    | 250   |
| T15        | 1      | 1       | 1      | 1       | 1         |        |         |        | ≥    | 20    |

Please cite this article as: A. Kumar, R.R. Barton. Controlled violation of temporal process constraints – Models, algorithms and results, Information Systems (2016), http://dx.doi.org/10.1016/j.is.2016.06.003
For each scenario the system calculates and displays the new violations on the right of the spreadsheet. The user can select a scenario and click the ‘check’ button to have the system produce any additional diagnosis of the scenario. Moreover, the user may select a scenario and click the accept button to force the system to accept a certain scenario. In this way a user can operate in an interactive mode with the system.

8. Discussion and related work

The approach presented above is general and extensible. Not only does it apply to structured processes with AND and XOR connectors, it can also be extended to new types of connectors such as inclusive and constrained ORs by defining the formal structural balance equations for them. Moreover, this approach applies to unstructured processes as well where split and join connectors are not well-nested or matched since we treat each connector separately. The MILP problem is NP-complete; however, very efficient solution techniques for it are known. The overhead of solving the MILP is incurred both at design time and also run time for each scenario where a deviation from the design time schedule occurs. More work is needed to determine the scalability of this approach as problem size increases.

The first efforts towards formally describing temporal patterns are due to Allen [2] who developed a general theory of action and time for reasoning about actions based on temporal logic. He also introduced relationships like before, equal, meets, overlaps, during, starts and finishes as a way of relating two or more time intervals, and then reasoning about them. However, the work of Allen was not done in the context of workflows. Temporal reasoning in the context of workflows is discussed in [5]. By far, the early efforts on introducing time into workflow systems were due to Marjanovic and Orlowska [21], Sadiq et al. [26] and Eder et al. [10,11]. The approach in [10,11] relies on ideas from project planning and critical path methods to determine various metrics like earliest start date, latest finish date, etc. for various activities. Zhao and Stohr [36] also developed a framework and algorithms for temporal workflow management in the context of a claims handling system based on turnaround time prediction, time allocation and task prioritization. They used reward functions to guide workers’ behavior. Timed coloured Petri net models have also been proposed as a way to model and analyze complex logistic systems [32]. They have applications in dynamic, real-time systems but they can be rather abstract from an end-user perspective.

Lanz et al. presented several time patterns (TP) that represent temporal constraints of time-aware processes [19,20] in the context of the ATAPIS project [1]. These patterns are in four groups: durations and time lags; restricting execution times; variability; and recurrent process elements. We showed in Section 6 that they can be represented using our approach. Time-BPMN is a proposal for incorporating temporal features and constraints into BPMN [12].

The approaches of [19,20] and others towards dealing with time-aware processes have relied on the conditional simple temporal networks (CSTN) as a representational technique [31]. These networks allow a mapping from time points at which observations are taken to propositional statements attached to nodes. These statements are checked for their truth values at the observation times and the corresponding actions at the nodes are performed if the statements are true. The propositions are Boolean combinations of simple range constraints. Techniques for checking such networks are discussed in the work of Combi et al. [7] where CSTN’s are also extended to CSTNU’s (CSTNs with uncertainty) which are more general CSTNs. In CSTNUs the uncertainly arises from the fact that some contingent edges become applicable only if a condition is satisfied.

CSTNs were developed in the context of planning problems and were later applied to workflows. Since the main idea is to partition the nodes in a CSTN based on the truth values of propositions, they do not always follow a workflow like structure. The idea of constraint violation and relaxation that we presented here does not exist in the context of CSTNs. Moreover, to the best our knowledge, extensions like overlap and iteration using special functions like max and min are also not present since they restrict the constraints to simple range constraints.

A related concept in the context of temporal workflows is that of dynamic controllability [6,17,22,31]. In this view, a temporal workflow consists of contingent links whose actual duration is determined by nature within a given range, and agent-controlled links whose actual value is under the control of and determined by an agent at execution time. The actual values of the durations of the contingent links are known only at run time. A CSTN or CSTNU network corresponding to a workflow process is dynamically controllable if there exists a viable agent strategy to successfully complete the execution of the workflow for all combinations of values of the contingent link durations. Algorithms for ensuring controllability are described in [7,17,22,31]. It is not possible to compare our approach directly with dynamic controllability because in our formulation there are no contingent links. All our task

Please cite this article as: A. Kumar, R.R. Barton, Controlled violation of temporal process constraints – Models, algorithms and results, Information Systems (2016), http://dx.doi.org/10.1016/j.is.2016.06.003
duration and inter-task duration ranges are determined based on, say, medical (or some other kind of) guidelines.

Health care is a particularly important application for management of temporal constraints. Workflows in a healthcare enterprise must respond and adapt in real-time [3] making temporal constraint management particularly important. A lot of effort in the ADEPT project [9] was made in the context of change and flexibility in healthcare. The work in [13] shows how to translate a time-annotated clinical pathway model into a temporal hierarchical task network (HTN). Temporal constraints on this network can be checked and managed using techniques from the HTN planning domain.

A rather different perspective for controlling violations is to proactively undertake certain measure or escalation steps when it is anticipated that the deadline for the completion of a process instance will be missed. In this line of research (e.g. [33]), escalation may imply performing a task in a different way, allowing less qualified people to do certain tasks, or making decisions based on incomplete data. Such measures can help reduce the number of missed deadlines. In [35] a time probability model is developed based on the historical logs of process executions. This model is used to develop an algorithm that can throw exceptions when deadlines are missed and make adjustments in the scheduling strategy to minimize further delays.

Another interesting constraint based approach for modeling clinical pathways is discussed in [34]. It considers resources and various scheduling patterns with setup costs and temporal constraints, but it does not use a process focus and does not provide a solution methodology. Stochastic approaches for modeling temporal durations are discussed in [25,27,28]. In [25] non-markovian stochastic distributions are used to represent the durations of an activity and run time predictions are made about completion times and risks of missing deadlines. The method in [27] combines queuing and simulation with statistical methods to analyze the performance of running processes described as colored Petri-nets. Compliance issues of temporal processes are discussed from a diagnostic and alignment perspective in [30] by matching actual log traces with a specified model and deviations are highlighted. Another approach to compliance based on rule-like temporal patterns is described in [4]. This work shows how BPMN processes can be extended to incorporate temporal patterns by using events and other BPMN structures for execution in cloud environments.

This review of related work shows that modeling, analysis and verification of temporal workflows is a fertile area of considerable research interest at the current time and a variety of approaches have been developed to address the problems in this area.

9. Conclusions

The focus of this work is on managing violations of temporal constraints in workflow systems in a controlled way. We presented a new approach that can check temporal consistency at both design and run times. A unique aspect of our approach is that it can allow controlled violation of constraints by allowing relaxation of some constraints and associating penalties with the violations. It was illustrated with a realistic example of a clinical workflow. However, the approach is general. We also showed that it can express a variety of temporal patterns and deal with changes in the process. Formal properties of the method and a proposal for implementation were given.

Future work should address implementation and testing of the approach, as well as integration into a workflow engine. Further investigation is also needed to compare our approach with the ones based on dynamic controllability and CSTN networks. More work is required in developing methods for associating the right penalties with constraints and optimal ways of recovering from violations. Compensation and substitution based methods may also be used for this purpose and costs can be included in the model as well. As an example, in a process running late, it may be possible to substitute an expensive procedure or diagnostic test that runs faster, say, in two days versus another less expensive one that takes four days.

Finally, activity durations, path choices, violation types, degrees and frequencies, and temporal patterns can all be statistically characterized. This would allow for a stochastic approach [25,27,28] to managing constraint violations by using the statistical characterization of paths and durations to allow optimal violation management in light of probabilistic inference for the yet-to-be completed part of the process.

Acknowledgment

This research was supported in part by the Smeal College of Business. Part of the work on this paper was done while the first author was a visitor at the School of Information Systems at Singapore Management University. He thanks the school for hosting him.

Appendix A

Linearizing constraints with minimum (min) and maximum (max) functions

Say, we have constraints like:

\[ Z \geq \text{Min}(A,B) \]  \text{or}  
\[ Z \leq \text{Max}(A,B). \]

(Note that \( Z \leq \text{Min}(A,B) \) or \( Z \geq \text{Max}(A,B) \) are trivial to linearize and so we don’t consider them.)

First, we can write

\[ \text{Min}(A,B) = \text{Max}(A,B) - |A - B| \]

Now, say the modulus of \( |A - B| = M \)

Then, we express the modulus \( M \) as

\[ M \geq A - B \]
\[ M \geq B - A \]

Next,

\[ Z \geq A - S^+ \]
\[ Z \geq B - S^+ \]

Where

\[ S^+ \geq A - B \]
\[ S^+ \geq B - A \]
\[ S^+, S^+ \geq 0 \]

Please cite this article as: A. Kumar, R.R. Barton, Controlled violation of temporal process constraints – Models, algorithms and results, Information Systems (2016), http://dx.doi.org/10.1016/j.is.2016.06.003
Similarly, we linearize $\text{Z} < \text{Max}(A, B)$ as follows:

\[
\begin{align*}
\text{Z} & \leq \text{Min}(A, B) + 1A - 1B \\
\text{Z} & \leq A + S^+ \\
\text{Z} & \geq B + S^+ \\
\text{Where} \\
S^+ & \geq A - B \\
S^+ & \geq B - A \\
S^-, S^+ & \geq 0
\end{align*}
\]

In this way, the max and min functions can be linearized.

**Appendix B**

Linearizing the rule: If $\text{TS}_1 > t_1$ and $\text{TF}_1 < t_2$, then $\text{TS}_2 > t_3$ and $\text{TF}_2 < t_4$.

Each expression on both the left and right sides of this rule represents a condition test, of the form $c > 0$ with a Boolean 0–1 value for false or true. This can be expressed in two linear constraints as ($M$ is a very large number):

\[
\begin{align*}
c & \leq M^X \\
-c & \leq M^X(1-X)
\end{align*}
\]

Thus, each of the four expressions can be converted into a binary variable, $X_1, X_2, X_3,$ and $X_4$ respectively. Thus, the rule becomes:

If $X_1$ and $X_2$ then $X_3$ and $X_4$

Next the pair of Boolean values $X_1$ and $X_2$ on the left are combined into a binary variable $L$ as:

\[
\begin{align*}
L & = X_1 + X_2 - 1 \\
L & \leq X_1 \\
L & \leq X_2
\end{align*}
\]

Similarly, $X_3$ and $X_4$ on the right are combined into a single binary variable $R$. Now the rule is:

If $L$ then $R$.

This can be expressed simply by a constraint as: $R \geq L$.

**References**


