2010 17th International Symposium on Temporal Representation and Reasoning

Towards Temporal Controllabilities for Workflow Schemata

Carlo Combi and Roberto Posenato

Department of Computer Science, University of Verona (Italy)
strada le Grazie 15, I-37134 Verona, Italy
e-mail: {carlo.combi,roberto.posenato}@univr.it

***Abstract*—The modelling and management of temporal constraints over business processes has received some attention in the past years. Recently, we have introduced and discussed the concept of *controllability* for workflow schemata modelling real world business processes: controllability, originally introduced in the AI community for temporal constraint networks, refers to the capability of executing a workflow for all possible durations of all tasks. In this paper, we first extend the execution strategy proposed by Morris, Muscettola and Vidal in the context of temporal constraint networks, to deal with the execution of business processes in a more suitable way. Then, we discuss and propose a new algorithm to deal with the (dynamic) controllability of an overall workflow schema, where several, possibly disjoint, execution paths are possible, due to the presence of alternative paths in the workflow schema. We show that the presence of several controllable alternative execution paths in a workflow schema does not guarantee that the overall workflow schema is controllable.**

***Keywords*-temporal conceptual workflow design; controllability; temporal constraint networks.**

I. Introduction

Organisations use workflow management systems (*WfMSs*) to streamline, automate, and manage business processes that depend on information systems and human resources [1]. Many business processes have different kinds of temporal restrictions such as a limited duration of single tasks or activity deadlines w.r.t. either the beginning of the workflow or a specific time point in the control flow. Generally, time violations lead to some form of exception handling, thus increasing the complexity of business process management. Therefore, a *WfMS* should provide the process manager with the necessary information about a process, its time restrictions, and its actual time requirements.

In the last decade some proposals have been made to represent and manage different temporal aspects of workflows as temporal constraints and deadlines both at design time and at run time [2], [3], [4], [5], [6]. Recently, we have introduced the concept of *controllability* for temporal workflows [6]: controllability has been proposed and studied in the AI community working on temporal constraint networks [7], [8] and, in the workflow context, it refers to the capability of executing a workflow for all possible durations of all tasks. Indeed, durations of tasks cannot be imposed by *WfMSs*: even though the minimum and the maximum durations for each task are known, the actual duration of a task is known only at run time, after the agent executed it. Therefore, checking controllability is stronger than verifying the consistency of workflow temporal constraints [2], [4]. In other words, each task duration cannot be imposed or decided by the *WfMS* that can only schedule all the tasks assuming that task durations respect their allowed ranges; hence, it is necessary to know if some schedule is possible before the start of the execution.

After having introduced and discussed in [6] temporal controllability for the basic workflow patterns, in this paper we (i) first extend the execution strategy proposed in the context of temporal constraint networks, to deal with the execution of business processes in a more suitable way; (ii) then, we focus on subtle and new issues arising when we consider the overall controllability of a workflow schema that may involve several different workflow execution paths. We discuss different kinds of controllability and propose a new algorithm to check the (different kinds of) controllability of an overall workflow schema. We then show that the presence of several controllable alternative execution paths in a workflow schema does not guarantee that the overall workflow schema is controllable.

II. Related Work

Two different research directions have to be considered: i) in the area of business process modelling and workflow management systems, some efforts have been devoted to the study of temporal aspects in representing and managing workflows (i.e., business processes) and their component activities (i.e., tasks); ii) in the AI-related area of temporal constraints, some studies addressed the issues related to constraint satisfiability when some temporal constraints are related to time distances between point-based events that are not under control, i.e., they happen freely according to the given constraints.

In the first research direction, in [2], Eder et al. introduce the Timed Workflow Graph (*TWG*) that represents temporal properties of tasks. *TWG* is a directed acyclic graph in which nodes are activities and oriented edges are control flows, allowing one to represent temporal constraints as upper and lower bounds for the end of a task w.r.t. either the start of the workflow or the end of any previous task. In [4],

1530-1311/10 $26.00 © 2010 IEEE
DOI 10.1109/TIME.2010.17

Marjanovic et al. define a conceptual model that distinguishes different temporal constraints for a workflow schema: "basic temporal constraint", "limited duration constraint", "deadline constraint" and "interdependent temporal constraint". A basic temporal constraint limits the expected duration of a single task. A limited duration constraint is an upper bound for the duration of the workflow execution. An interdependent temporal constraint limits the time distance between two tasks in a workflow schema. In [5], Combi et al. propose a temporal conceptual workflow model that enhances the expressiveness of previous proposals in representing temporal constraints, such as those related to tasks and task connectors.

The model by Bettini et al. [3] is quite different from the previous ones and can be considered as the first attempt to merge the afore-mentioned research directions on temporal workflow models and on temporal constraint networks. In [3], a workflow graph represents any task of a workflow by two nodes, corresponding to the starting and the ending instant of the considered task, respectively. Every edge in the workflow graph represents the temporal distance between two nodes: any edge label is an interval representing the allowed time distances between the connected nodes. Moreover, the authors introduce the concept of *free schedule*: a schedule is free when it is possible to statically fix the start times of all tasks without constraining their durations and satisfying all the given constraints, before the beginning of the execution. A polynomial algorithm ($O(n^4)$ where n is the number of nodes) is then provided to check the existence of a free schedule.

Free schedules resemble the concept of controllability, that has been mainly investigated by Morris, Muscettola, and Vidal [7], [8], in the AI area of temporal constraint networks for planning. Assuming that a temporal planner has to manage the likely uncertainty about the duration of processes, Vidal et al. propose an extension of the Simple Temporal Network, the Simple Temporal Network With Uncertainty (*STNU*), where edges, i.e., constraints, are divided into two classes: *contingent links* and *requirement links*. Contingent links represent processes of uncertain duration, where finish timepoints (i.e., STNU nodes) are decided by Nature within the limits imposed by the bounds defined on the contingent links. Requirement links represent all the other processes whose finish timepoints are controlled by the agents that execute processes. Informally, *Controllability* refers to the capability of specifying all the timepoints controlled by agents, satisfying all the requirement and contingent links. In particular, *dynamic controllability* ensures that it is possible to specify at runtime the timepoints controlled by agents only by knowing the duration of the already happened contingent links, without preventing any possible duration of the future contingent links [7]. Several algorithms have been proposed to check the dynamic controllability of a constraint network [8]; eventually, Morris showed that in the framework of *STNU* the checking controllability algorithm is polynomial, i.e., $O(n^4)$, w.r.t. the number of *STNU* nodes [9]. In [10], Hunsberger highlights some issues in the approach proposed by Morris and Muscettola and proposes a stronger definition of dynamic execution strategies that fixes these problems and puts the checking algorithm on a more solid theoretical foundation.

III. Background

In our model, the specification of a (structured) workflow is given by a *workflow schema*, a directed graph (also called workflow graph) where nodes correspond to *activities* and edges represent *control flows* that define activity dependencies on the order of execution [5], [6]. There are two different types of activity: *task* and *connector*. *Tasks* represent elementary work units that will be executed by external agents. They are graphically represented by a rounded box and each task has, at least, the mandatory attribute *duration* specifying the allowed temporal spans for its execution. The value (i.e., the range) of *duration* of a task cannot be modified by the *WfMS*. *Connectors* represent internal activities executed by the *WfMS* to achieve a correct and coordinated execution of tasks. They are graphically represented by diamonds and each of them has the mandatory attribute *duration* specifying the temporal spans allowed to the *WfMS* for executing it. The value of a connector *duration* can be modified at run time to guarantee the right coordination, and the effective duration is decided by the *WfMS*. There are two kinds of connectors: *split* and *join*. *Split* connectors are nodes with one incoming edge and two or more outgoing edges: after the execution of the predecessor, (possibly) several successors have to be considered for the execution. The set of nodes that can start their execution is given by the kind of split connector. A *split* connector can be: *Total*, *Alternative* or *Conditional*. *Join* connectors are nodes with two or more incoming edges and one outgoing edge only. A *join* connector can be either *And* or *Or*. *Control flow* is an oriented edge that connects two activities: the first activity (*predecessor* of the second one) must be finished before starting the execution of the second one. Every edge has a temporal property, *delay*, that denotes the allowed times that can be spent by the *WfMS* for possibly delaying the enactment of the second activity according to the given temporal constraints. Each workflow can contain exactly one Start node and one End node without any temporal attribute, graphically represented by a circle with one ingoing/outgoing edge, respectively.

Due to conditional and alternative flows, not all the cases (i.e., executions) of a workflow schema perform exactly the same set of tasks. We group workflow cases into *workflow paths* (*wf-paths*) in accordance with the activities actually executed. Therefore, a *wf-path* can be regarded as a workflow subgraph in which all alternative or conditional connectors have exactly one successor.

Allowed durations/delays are expressed by ranges like [MinD, MaxD] Granularity where $0 \leq \mathsf{MinD} \leq \mathsf{MaxD} \leq \infty$ and

Granularity stands for the time unit used to express the given constraint. If the workflow designer does not set a duration, we assume the duration value to be $[1,+\infty]$ MinG, where MinG is the finest granularity managed by the *WfMS*. We do not admit 0 as value of MinD for durations/delays at MinG to underline that no activity can be executed without time consumption. Moreover, we assume min(ute) as MinG when it is not explicit in a range specification. For sake of simplicity, here we will not consider temporal constraints expressed at different granularity and related conversion issues.

Besides the basic temporal constraints, our conceptual model allows the expression of several other kinds of temporal constraints as the *relative* constraints. A *relative constraint* limits the time distance (duration) between the starting/ending instants of two non-consecutive workflow activities expressed according to the following pattern: $\langle \mathsf{I_F} \rangle [\mathsf{MinD}, \mathsf{MaxD}] \langle \mathsf{I_S} \rangle$ Granularity, where (i) $\langle \mathsf{I_F} \rangle$ marks which instant of the First activity to use ($\langle \mathsf{I_F} \rangle = \mathsf{S}_{\langle \mathsf{activity} \rangle} \mid \mathsf{E}_{\langle \mathsf{activity} \rangle}$ as the starting/ending execution instant, respectively; the subscript can be omitted if it is clear from the context.); (ii) $\langle \mathsf{I_S} \rangle$ marks the instant for the Second activity in the same way; (iii) [MinD, MaxD] Granularity represents the allowed range for the time distance between the two instants $\langle \mathsf{I_F} \rangle$ and $\langle \mathsf{I_S} \rangle$. We assume that $-\infty \leq \mathsf{MinD} \leq \mathsf{MaxD} \leq \infty$.

In general, a *WfMS performs* a *wf-path* assigning tasks to agents and executing connectors, observing structural constrains and temporal ones. We say that a workflow schema is *controllable* if the *WfMS* is able to perform any *wf-path* satisfying all relative constraints, all delays, all connector durations without setting (allowed) task durations involved in the *wf-path*. Analysing the structure of workflow schemata, it is straightforward to verify that (i) if a workflow schema does not contain any total connector, then each *wf-path* is represented as one graph-path *(sequential path)* and (ii) if the workflow schema contains at least one total connector, then at least one *wf-path* is represented as two or more graph-paths *(parallel paths)*. The problem of controllability checking arises when there is at least one relative constraint that involves two or more tasks. In [6] we discussed how to check the controllability of all patterns that can be present in sequential paths or in parallel ones. Here, we summarise the results in Fig. 1 and Fig. 2.

Fig. 1 shows two fundamental sequential patterns and the most frequently derived one. In pattern (a) the composition of the task duration and of the delay has to comply with the relative constraint. Duration of task T1 cannot be modified: the *WfMS* can only decide the duration of delay, after T1 is executed. In order to verify whether it is possible to guarantee that the relative constraint can be satisfied for every possible T1 duration, it is sufficient to verify whether the range $[p-y_1, q-x_1] \subseteq [u,v]$. Pattern (b) is similar to (a), but the task duration is still unknown when the *WfMS* has to decide the duration of the delay. In order to guarantee that

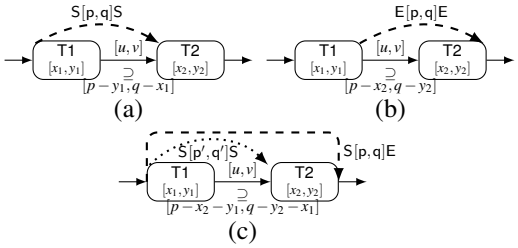

Figure 1. Three sequential patterns with a relative constraint. Patterns are significant whether $0 \leq p \leq q$. In (c) the relative constraint $\mathsf{S}[\mathsf{p}',\mathsf{q}']\mathsf{S}$ (dotted) is induced by $\mathsf{S}[\mathsf{p},\mathsf{q}]\mathsf{E}$

the relative constraint can be satisfied for every possible T2 duration, it is necessary to impose a stronger condition on the delay range: the restricted valid range is $[p-x_2, q-y_2] \subseteq [u,v]$. Pattern (c) is the common propagation of a relative constraint. The relative constraint has the form $\mathsf{S_{T1}}[\mathsf{p},\mathsf{q}]\mathsf{E_{T2}}$ and we want to analyse how it is propagated to the edge T1-T2. The controllability check can be done in two steps. In the first step, a new relative constraint between $\mathsf{S_{T1}}$ and $\mathsf{S_{T2}}$ is defined, and its temporal range is determined by applying the rule of pattern (b). If the induced relative constraint is not empty, in the second step, the controllability of T1 duration and the edge delay w.r.t. the new constraint is verified by applying pattern (a). If all steps are successfully performed, the pattern is controllable.

In general, given a *wf-path*, the controllability analysis requires checking the controllability of all temporal constraints (i.e., between any pair of nodes). To determine all temporal constraints, it is sufficient to transform the *wf-path* into the equivalent instance of Simple Temporal Problem (*STP*) and apply an *all-pairs shortest path* algorithm, as Floyd-Warshall [11]: the consistency check determines the minimal satisfiable temporal constraint between any pair of nodes of the network if the original network admits them. There are four possible cases w.r.t. the result of *STP* consistency check: (i) the consistency check fails, (ii) the new constraint ranges are equal to the corresponding old ones, (iii) at least one task range has been restricted, and (iv) only non-task range(s) has (have) been restricted. In (i) and (iii) cases, the *wf-path* is not controllable, in (ii) case the controllability analysis is completed and the *wf-path* is called *pseudo-controllable*, and in (iv) case it is necessary to check the controllability of new ranges and, then, to apply the *all-pairs shortest path* algorithm again, in order to verify if the stable state has been reached.

Regarding parallel paths, there are four basic parallel patterns, each containing a relative constraint as in Fig. 2. In pattern (d) the composition of duration of T1 and of the relative constraint results in the derived constraint $\mathsf{A}[\mathsf{u},\mathsf{v}]\mathsf{E_{T1}} = \mathsf{A}[\mathsf{x_1}-\mathsf{q},\mathsf{y_1}-\mathsf{p}]\mathsf{E_{T1}}$ as a generalisation of sequential pattern (a), although here the direction of $\mathsf{A}[\mathsf{u},\mathsf{v}]\mathsf{E_{T1}}$ is reversed. Pattern (e) is the most interesting one. Due to lack of space, we cannot report all the analysis made in [6]. Here we note only that: (i) if $q < 0$ then it is sufficient to choose

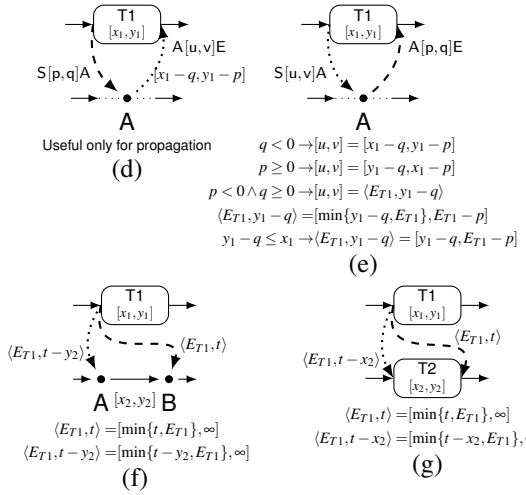

Figure 2. Four parallel patterns with a relative constraint. We remember here that $p \leq q$. The dotted edges are induced relative constraints by the composition of the given relative constraint and the T1 duration. For sake of simplicity, we put A in the labels of relative constraints, as A could represent either a starting or an ending instant of an activity. In (f) and (g) the relative constraints are *wait* constraints

a suitable value in the range $[x_1 - q, y_1 - p]$ (depending on the T1 duration) as delay of the edge T1-A to control the pattern, (ii) if $p \geq 0$, it is sufficient to fix the delay of T1-A to be $[y_1 - q, x_1 - p]$ to have the controllability, (iii) if $p < 0$ and $q \geq 0$, it is not possible to set a single range to guarantee the controllability but it is necessary to set a new constraint (*wait* constraint) between the start of T1 and A that is conditioned by the end of T1. The *wait* constraint has the special label $\langle E_{T1}, y_1 - q \rangle$ that means: A could occur either when (1) "T1 has ended (and within $|p|$ time units)" or when (2) "$y_1 - q$ time units have elapsed since the start of T1 and T1 has not yet finished" (if A does not occur when condition (2) holds, the following end of T1 will trigger condition (1)). Sometimes the *wait* constraint can be simplified: if $(y_1 - q) \leq x_1$, then a lower bound can be set because condition (2) is always verified before the end of T1: so the constraint can be represented as $[y_1 - q, E_{T1} + |p|]$.

IV. Extending the Wait Semantics for Wf-path

In the proposal by Morris, Muscettola, and Vidal, the semantics of the wait constraint is that the "waiting" time point must be executed as soon as the wait condition holds (i.e., either a suitable number of time units has elapsed since the start of the "waited" task or the "waited" task ended). In the workflow domain, "waiting" time points could correspond to the end of the execution of some internal workflow activities, such as the evaluation of conditions for routing or assignments of tasks to agents, that could require some time: thus, it becomes interesting to state explicitly the maximum delay the *WfMS* has at disposal for executing the "waiting" time point after that the wait condition became true. In order to make more clear the Morris' wait constraint $\langle E_{T1}, y_1 - q \rangle$, it is well-timed to consider the upper bound

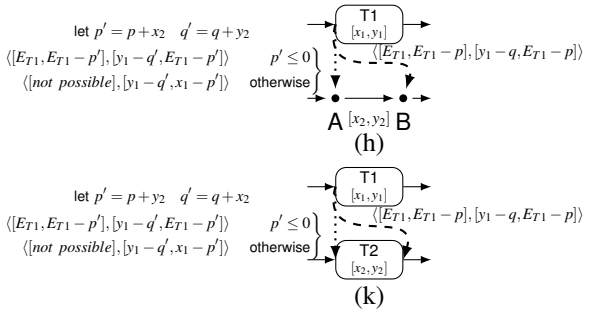

Figure 3. Augmented wait constraint and its regression through a edge or connector (h) and through a task (k).

too:

$$\langle [E_{T1}, E_{T1} - p], [y_1 - q, E_{T1} - p] \rangle$$

This augmented wait constraint has to be translated in one of the following constraints in order to check the pseudo-controllability:

$$[y_1 - q, y_1 - p] \text{ if } y_1 - q \leq x_1$$
$$[x_1, y_1 - p] \text{ otherwise}$$

Morris et al. proposed a method, called *regression*, to propagate wait constraints to other links. In [6], we discussed how to adapt such regression to the context of workflow patterns, as summarised in Fig. 2-(f)-(g), for the regression through a connector and through a task, respectively.

We want to propose a regression for the augmented wait constraint in order to allow the system to start an event under a wait not just at the first possible time point but at one time point of a range. The overall idea is to propagate the upper bound $(E_{T1} - p)$ of the augmented wait constraint through the regression.

As for the regression through a connector, depicted in Fig. 2-(f), the wait constraint between the start of the task and point *B* becomes $\langle [E_{T1}, E_{T1} - p], [y_1 - q, E_{T1} - p] \rangle$ while the wait between the start of the task and point *A* becomes $\langle [E_{T1} - x_2, E_{T1} - p - x_2], [y_1 - q - y_2, E_{T1} - p - x_2] \rangle$. Since $x_2 \geq 0$, $E_{T1} - x_2$ is a non sense because we cannot consider a time instant before the unknown end of the task, and thus it is necessary to increase the bound to E_{T1}. For the same reason, if $p' = p + x_2 > 0$, we cannot consider the upper bound $E_{T1} - p'$: in that case we have to consider $x_1 - p'$ instead of $E_{T1} - p'$ since x_1 is the minimum value that E_{T1} can assume. It is important to underline that $y_1 - q - y_2 \leq x_1 - p'$, otherwise there would be a negative cycle on $\mathsf{S_{T1}}$. Therefore, to set the upper bound of the augmented wait constraint after the regression, we have to consider only one condition: if $p' = p + x_2 \leq 0$, then the upper bound is $E_{T1} - p'$, otherwise it is $x_1 - p'$. Fig. 3-(h) shows the new representation of this wait constraint.

As for the regression through a task, depicted in Fig. 2-(g), the wait constraint between the start of the task T1 and the end of task T2 becomes $\langle [E_{T1}, E_{T1} - p], [y_1 - q, E_{T1} - p] \rangle$ while the wait between the start of the task T1 and the start of T2

becomes $\langle [E_{T1}-y_2, E_{T1}-p-y_2], [y_1-q-x_2, E_{T1}-p-y_2] \rangle$ in order to consider the possibility that T2 lasts its maximum duration. Since $y_2 \geq 0$, $E_{T1}-y_2$ is a non sense because we cannot consider time instant before the unknown end of the task, therefore it is necessary to increase the bound to E_{T1}. For the same reason, if $p' = p + y_2 > 0$, we cannot consider the upper bound $E_{T1}-p'$: in that case we have to consider $x_1 - p'$ instead of $E_{T1}-p'$. Fig. 3-(k) shows the new representation of this wait constraint.

V. Controllability of Workflow Schemata

Now we can analyse some issues when we need to verify the controllability of the overall workflow schema. In this case we have consider all the possible *wf-paths* together. As we will discuss in detail, it turns out that the controllability of each *wf-path* in isolation does not guarantee the controllability of the overall schema.

Before we discuss the different kinds of controllability, we introduce a preliminary concept, i.e., that of *prefix* of an activity/edge, to group together *wf-paths* having some common set of initial activities and edges.

Definition 5.1 (activity/edge prefix): A *prefix* of a given activity/edge y is the set of all the *wf-paths* that have the same successor for each alternative or conditional connector that *precedes* y. An activity x precedes an activity y if either it belongs to the predecessors of y or precedes an activity of this set.

Hereinafter a prefix of a given activity/edge y is represented by the path notation specifying the common part that *wf-paths* of the prefix share. This way, it is straightforward to define a partial order $\preceq$ among prefixes according to the lexicographical order induced by this notation. For example, considering prefixes T1-T3 and T1-T3-T4, it holds T1-T3 $\preceq$ T1-T3-T4, as depicted in Fig. 8. It is worth noting that for any prefixes α, β s.t. $\alpha \preceq \beta$ the *wf-paths* of β belong to α too, i.e., $\alpha \supseteq \beta$.

We distinguish three kinds of controllability for a workflow schema: *strong, weak*, and *history-dependent controllability*.

Strong controllability refers to the fact that it is possible to derive a common range for durations of each connector/edge ensuring the controllability of each *wf-path* of the workflow schema. In this case any derived duration range does not prevent any possible future evolution of the workflow execution, i.e., the duration range guarantees the controllability of all the *wf-paths* of any prefix of the considered connector/edge. Moreover, the derived duration range does not depend on the specific execution flow followed to reach the considered connector/edge (i.e., the history): indeed, the range is common to all the different prefixes of the considered connector/edge. Thus, strong controllability requires two different properties for each connector/edge: i) the controllability for any possible future execution flow, and ii) the independence of controllability from the past execution flow.

Function controllabilityCheck(G)

Input: G: workflow graph to analyse.
Output: the controllability of G and a possibly new set of ranges for activities/delays.
// Initial *wf-paths* controllability check
foreach (*wf-path* $p \in G$) **do**
  $R =$ ranges in p;
  $(status, R) =$pathControllabilityCheck(p, R);
  if ($status ==$ *'Non Controllable Path'*) **then**
    return *('Non Controllable Wf', ∅)*
end
// All *wf-paths* are controll., determine the kind of controll.
$(status, R) =$controllabilityCheck$(G,$ Start$)$;
if ($status ==$*'Strongly Controllable'*) **then**
  return *('Strongly Controllable Wf', R);*
// Determine the ✕ join connectors to analyse
✕JoinSet = $\{x \mid x$ is the last ✕ Join in a *wf-path*$\}$;
foreach ($x \in$ ✕*JoinSet*) **do**
  $(status, R) =$controllabilityCheck(G, x);
  if ($status ==$*'Strongly Controllable'*) **then**
    return *('History-Dependently Controllable Wf', R)*
end
return *('Weakly Controllable Wf', ∅);*

Figure 4. Algorithm to check the controllability of a workflow graph

While the first property is non-negotiable, the second one could be relaxed allowing a temporal range that depends on the past execution but without affecting the controllable execution of the remaining workflow schema. This leads us to the concept of *history-dependent controllability*: a workflow schema is history-dependently controllable if each connector/edge has (possibly) different duration ranges for different prefixes: for each prefix, the given duration range is common to all the *wf-paths* of that prefix. In this case the duration range of a connector/edge could depend on the executed tasks (i.e., the history), but does not prevent the *WfMS* to execute any possible future task.

If the property of being able to follow any possible future execution of the workflow schema is not guaranteed, we could have a workflow schema composed by several *wf-paths* controllable in isolation where some controllable execution could be prevented at run time: we call this property *weak controllability*.

The algorithm *controllabilityCheck(G)* we propose determines whether a given workflow schema G is controllable, its kind of controllability, and duration ranges for connectors/edges (duration ranges of tasks must be leaved unchanged). Figure 4 shows the pseudo-code of the proposed algorithm: the algorithm verifies that each single *wf-path* is controllable in isolation and then executes *controllabilityCheck(G, Start)*, to verify whether there is a common range for each connector/edge for all the *wf-paths*. If it is the case, then the workflow schema is strongly controllable and the derived ranges have to be used at run time. Otherwise, the algorithm verifies whether the workflow schema is history-dependently controllable: it considers the (possibly several) last Or-*join*s of the workflow schema. Or-

Function controllabilityCheck($G, root$)

Input: G: a workflow graph; *root*: activity from which to start the analysis
Output: the controllability type of G and a new set of ranges for activities and delays

$P =$ set of all prefixes of *root*;
// R_p= array of ranges of activities/edges present in prefix p
foreach $prefix \in P$ **do** // Initialise all R_{prefix}
  Build R_{prefix} considering original temporal ranges in G;
end
do
  foreach $\{prefix \mid R_{prefix} \neq null\}$ **do**
    $R'_{prefix} = R_{prefix}$;// Save last found solution
  end
  foreach $\{prefix \in P\}$ **do**
    // Check if the prefix is strongly controllable
    $(status, R_{prefix})$=buildControllableRanges($prefix, R_{prefix}$);
    if *(status* $\neq$ *'Controllable Ranges Found')* **then**
      // It is necessary to evaluate another *root*
      return *('Non Strongly Controllable', ∅);*
  end
  foreach $\{a \mid a$ *is a conn./edge* $\wedge a$ *precedes root*$\}$ **do**
    // Determine the controll. range for a among prefixes
    $P' = \{p' \mid p'$ is prefix of $a\}$;
    foreach $prefix' \in P'$ **do**
      $R_{prefix'}[a]$=$\bigcap$of all a ranges in *wf-paths*$\in prefix'$;
      if $(R_{prefix'}[a] == \emptyset)$ **then**
        return *('Non Strongly Controllable', ∅);*
      foreach $\{prefix'' \mid prefix' \preceq prefix''\}$ **do**
        /* Propagate the new range to all prefixes greater than *prefix'* */
        $R_{prefix''}[a] = R_{prefix'}[a]$;
      end
    end
  end
while $(\exists$ *prefix of activity/edge* $b \mid R_{prefix}[b] \neq R'_{prefix}[b])$;
return *('Strongly Controllable',* $\cup_{prefix} R_{prefix}$*);*

Figure 5. Algorithm to check the controllability of a prefix

*join*s are considered as they are responsible of varying the number of prefixes (i.e., histories) of all activities following them. Starting from the last Or-*join*s, indeed, we verify that for each connector/edge of each prefix there is a suitable range not preventing any possible future execution path. This range could be different according to the considered prefix and ranges related to different prefixes could even be disjoint. To do this check, *controllabilityCheck(G)* uses *controllabilityCheck*($G, root$).

The algorithm *controllabilityCheck*($G, root$), shown in Fig. 5, firstly determines the prefixes of passed activity *root*. For each prefix, it verifies the strong controllability: if a prefix is strongly controllable, then every connector/edge in the prefix has a duration range common to all the *wf-paths* of the prefix itself. If all the prefixes are strongly controllable, then it is possible to verify whether every connector/edge preceding *root* has a duration range, possibly depending on the previously executed activities, that does not prevent any future execution path: in other words, we verify whether it is possible for these connectors/edges

Function buildControllableRanges(S, R)

Input: S: set of *wf-paths*; R: collection of connector/edge temporal ranges
Output: the status and a new set of temporal ranges for connectors/edges

do
  $R' = R$;
  foreach *connector c present into any wf-paths* **do**
    c range = $\bigcap$ of c ranges present into *wf-paths*;
  end
  foreach *edge d present into any wf-paths* **do**
    d range = $\bigcap$ of d ranges present into *wf-paths*;
  end
  $R =$ collection of these new ranges;
  foreach *wf-path* $p \in S$ **do**
    $(status, R)$ = pathControllabilityCheck(p, R);
    if *(status == 'Non Controllable Path')* **then**
      return *('Non Controllable Path Found', ∅)*
  end
  if *(at least one range in R is empty)* **then**
    return *('Empty Range Found', ∅)*
while *(at least one range in R has changed w.r.t.* R'*)*;
return *('Controllable Ranges Found', R)*

Figure 6. Algorithm to check the controllability of a set of *wf-paths*

Function pathControllabilityCheck(p, R)

Input: p: *wf-path*, R: set of connector/edge temporal ranges
Output: the controllability status and, possibly, the new set of temporal ranges for connectors and delays

do
  $R' = R$;
  Execute Floyd-Warshall on the distance graph corresponding to p with ranges in R;
  if *(p is not consistent)* **then**
    return *('Non Controllable Path', ∅)*
  Determine new ranges in R applying all possible constraint reductions and wait regressions on p;
  if *(*$\exists$ *empty range* $\in R \vee \exists$ *task range modified)* **then**
    return *('Non Controllable Path', ∅)*
while *(at least one range in R has changed w.r.t.* R'*)*;
return *('Controllable Path', R)*

Figure 7. Algorithm to check the controllability of a *wf-path*

to derive a common duration range for each of their prefixes. If it is possible, we are able to conclude that the workflow schema is history-dependently controllable. The derivation of these duration ranges may require to execute several times *buildControllableRanges*($prefix, R_p$) (depicted in Fig. 6) until a fixpoint is reached: indeed, the change of a duration range for a connector/edge may produce other range changes for *wf-paths* containing the considered connector/edge. These changes, in their turn, require that the controllability of modified *wf-paths* has to be checked again (through *pathControllabilityCheck*(p, R) depicted in Fig. 7) and it may induce further changes in the duration ranges of connectors/edges.

If the check of history-dependent controllability fails, then the workflow schema is weakly controllable: this kind of controllability is not completely appreciable by the *WfMS*, as it does not guarantee that any execution of the workflow

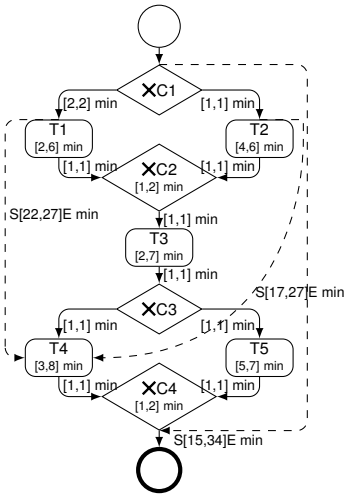

Figure 8. An example of workflow schema. Duration ranges for XC1 and XC3 are left to the default values of $[1,+\infty]$

schema will terminate correctly: for example, the duration allowed and used for a connector could led the *WfMS* to exclude a possible *wf-path* that results, instead, to be necessary to conclude the case in the right way.

A. Examples

To give a flavour of how our algorithms work, let us consider the workflow schema depicted in Fig. 8. It is composed by five tasks and by four connectors. The connectors determine four different *wf-paths*: T1-T3-T4, T1-T3-T5, T2-T3-T4, and T2-T3-T5.

It is possible to verify that the workflow schema is not strongly controllable: indeed, as it will be clear at the end of this discussion, it is not possible to find a common duration range for connector XC3 that guarantees the controllability for each *wf-path*. Therefore, the algorithm has to verify the history-dependent controllability starting with the analysis of prefixes of the last connector of workflow schema, i.e., XC4. XC4 has four prefixes, each of them containing a single *wf-path*. Fig. 9 shows the derivation of duration ranges for the four prefixes, firstly. With these new ranges, the algorithm determines the ranges for connectors and edges preceding XC4 w.r.t. their own prefixes. In particular, it is interesting to observe the derivation of duration ranges for XC3 prefixes and XC1 ones (connectors XC3 and XC1 are the only ones having different duration ranges on different *wf-paths*). Fig. 9 shows then the derived ranges for XC3 considering prefixes T1-T3 and T2-T3, respectively.

XC3 has a common range for each of its prefixes, $[1,10]$min for T1-T3 and $[1,8]$min for T2-T3, guaranteeing the controllability. Moving up to connector XC1, there is only one prefix (Start in Fig. 9) containing all the *wf-paths*; the algorithm verifies that the intersection among the XC1 duration ranges of all the prefixes of XC4 considering the new derived range for XC3 assures the controllability of each *wf-path* (and there are no changes in the previously derived ranges). Having reached the start of the workflow schema, we can say that the overall workflow schema is history-dependently controllable.

VI. Discussion and Conclusions

In this paper we discussed the issues arising when we deal with controllability of workflow schemata; in particular, we introduced three different kinds of controllability for a workflow schema and proposed an algorithm to check them. As for the complexity of the algorithm, it results that the overall complexity is exponential w.r.t. the order of the workflow graph. In more details, the time complexity of *pathControllabilityCheck(p,R)* is $O(n^3)$ (time complexity of Floyd-Warshall algorithm) times the maximum number of iterations of the do-while cycle. Such maximum number, even though in our experimental tests it results to be around the number 4, is limited by the maximum extension of temporal ranges present into path p (*MaxRange*). Therefore, the time complexity of *pathControllabilityCheck(p,R)* is pseudo-polynomial w.r.t. the size of p, and *MaxRange*, i.e., $O(n^3 MaxRange)$. Similarly, since *buildControllableRanges(S,R)* makes a limited number of operations on the results of path controllability check, its time complexity can be determined by multiplying the time complexity of *pathControllabilityCheck(p,R)* by the number of possible *wf-paths* of S and by maximum number of iterations of the do-while cycle: $O(n^3\,|S|\,MaxRange)$. As regards *controllabilityCheck(G,root)*, the complexity can be estimated by multiplying the complexity of *buildControllableRanges* by the number of possible prefixes of *root* and by the number of iterations of the do-while cycle. The number K_{root} of possible prefixes of *root* is given by counting all possible flows from Start to *root*. As in the previous case, the number of iterations of the do-while cycle is limited by the maximum extension of temporal ranges present into *wf-paths* of prefixes of *root* (*MaxRange*). Thus the overall complexity of *controllabilityCheck(G,root)* can be expressed as $O(K_{root}\,|S_{root}|\,n^3\,MaxRange^2)$ where S_{root} is the average cardinality of prefixes of *root*, observing that the product $K_{root}\,|S_{root}|$ corresponds to the total number of *wf-paths* of all prefixes of *root*. Denoting by w the number of *wf-paths* of the graph, it holds that $K_{root}\,|S_{root}| \leq w$ and so the *controllabilityCheck(G,root)* complexity can be approximated as $O(w\,n^3\,MaxRange^2)$. Finally, the time complexity of *controllabilityCheck(G)* is given by the order of complexity of *controllabilityCheck(G,root)* multiplied by the dimension of XJoinSet. An estimation of the (worst-case) dimension of XJoinSet is given assuming that the last Xjoin connectors are on a parallel path (each on a different branch) and their degree is 2: in this case the dimension of XJoinSet is less than the half of the number of *wf-paths* of the graph. Therefore the time complexity is $O(w^2\,n^3\,MaxRange^2)$. It's worth to note that the number of *wf-paths* $w \leq 2^c(a+1)^j$, where c is the total number of conditional connectors and $a = \max\{|a_l \star|\}$

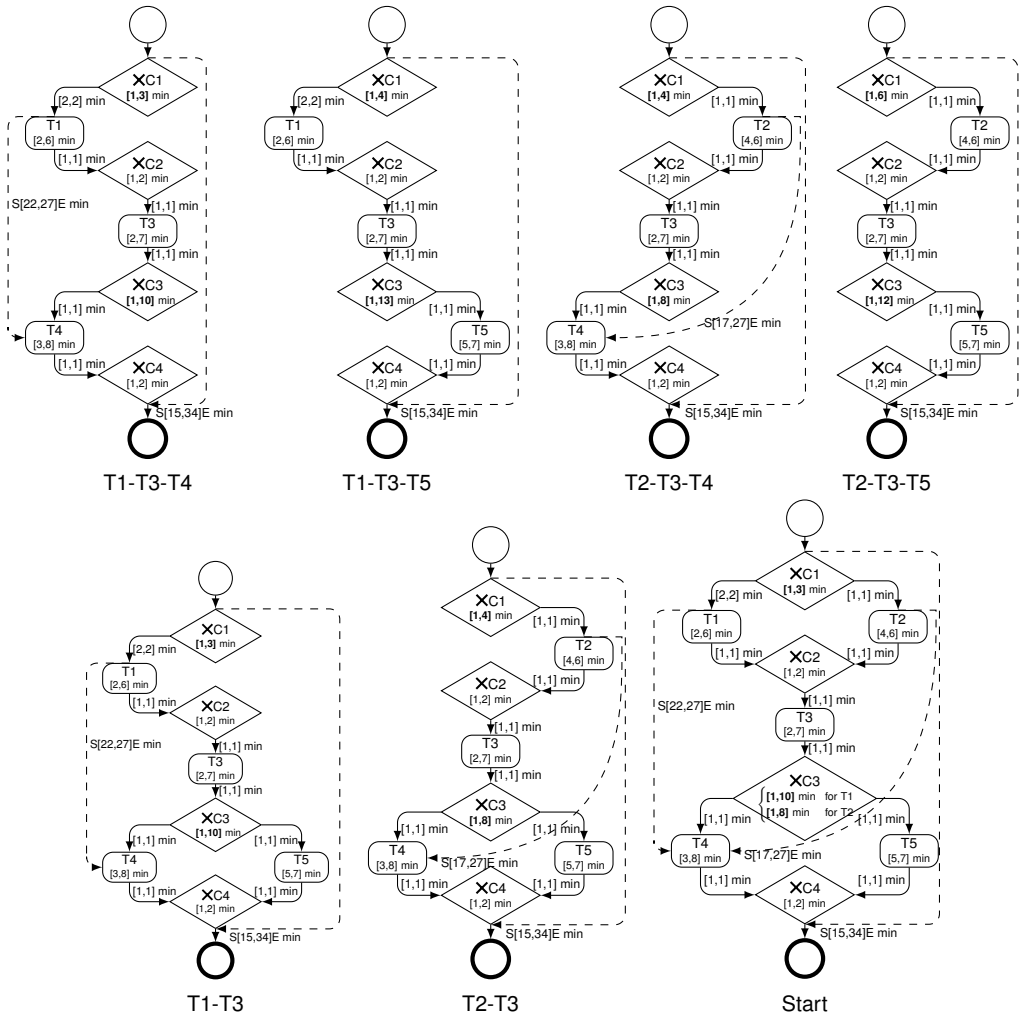

Figure 9. ✕C4 prefixes (T1-T3-T4, T1-T3-T5, T1-T2-T4 and T1-T2-T5 ones), ✕C3 prefixes (T1-T3 and T2-T3 ones) and ✕C1 prefix (Start prefix) with controllable ranges. The bold ranges are modified by the controllability algorithm w.r.t. the original ones. ✕C2 has the same prefixes of ✕C3

is the maximum among the out degrees of the alternative connectors.

REFERENCES

[1] Object Management Group (OMG), "Business process definition metamodel (bpdm), Beta 1," http://www.omg.org, 2007.

[2] J. Eder, W. Gruber, and E. Panagos, "Temporal modeling of workflows with conditional execution paths," in *DEXA*, ser. LNCS, M. T. Ibrahim, J. Küng, and N. Revell, Eds., vol. 1873. Springer–Verlag, 2000, pp. 243–253.

[3] C. Bettini, X. S. Wang, and S. Jajodia, "Temporal reasoning in workflow systems," *Distributed and Parallel Databases*, vol. 11, no. 3, pp. 269–306, 2002.

[4] O. Marjanovic and M. E. Orlowska, "On modeling and verification of temporal constraints in production workflows," *Knowl. Inf. Syst.*, vol. 1, no. 2, pp. 157–192, 1999.

[5] C. Combi, M. Gozzi, J. M. Juárez, B. Oliboni, and G. Pozzi, "Conceptual modeling of temporal clinical workflows," in *TIME*. IEEE, 2007, pp. 70–81.

[6] C. Combi and R. Posenato, "Controllability in temporal conceptual workflow schemata," in *BPM*, ser. LNCS, U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, Eds., vol. 5701. Springer–Verlag, 2009, pp. 64–79.

[7] T. Vidal and H. Fargier, "Handling contingency in temporal constraint networks: from consistency to controllabilities," *J. Exp. Theor. AI*, vol. 11, no. 1, pp. 23–45, 1999.

[8] P. H. Morris and N. Muscettola, "Temporal dynamic controllability revisited," in *AAAI*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press, 2005, pp. 1193–1198.

[9] P. Morris, "A structural characterization of temporal dynamic controllability," in *CP*, ser. LNCS, F. Benhamou, Ed., vol. 4204. Springer–Verlag, 2006, pp. 375–389.

[10] L. Hunsberger, "Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies," in *TIME*, C. Lutz and J.-F. Raskin, Eds. IEEE, 2009, pp. 155–162.

[11] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artif. Intell.*, vol. 49, no. 1-3, pp. 61–95, 1991.