# Conceptual Modeling of Flexible Temporal Workflows

CARLO COMBI, University of Verona
MATTEO GOZZI
ROBERTO POSENATO, University of Verona
GIUSEPPE POZZI, Politecnico di Milano

19

Workflow technology has emerged as one of the leading technologies in modeling, redesigning, and executing business processes. The management of temporal aspects in the definition of a workflow process has been considered only recently in the literature. Currently available Workflow Management Systems (*WfMS*) and research prototypes offer a very limited support for the definition, detection, and management of temporal constraints over business processes. In this article, we propose a new advanced workflow conceptual model for expressing time constraints in business processes and we present a general technique to check different levels of temporal consistency for workflow schemata at process design time: since a time constraint can be satisfied in different ways, we propose a classification of temporal workflows according to the way time constraints are satisfied. Such classification can be used to successfully manage flexible workflows at runtime.

Categories and Subject Descriptors: H.4.1 [**Information Systems Applications**]: Office Automation—*Workflow management*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms: Algorithms, Design, Management, Verification

Additional Key Words and Phrases: Temporal workflow, temporal consistency, flexible workflows

**ACM Reference Format:**
Combi, C., Gozzi, M., Posenato, R., and Pozzi, G. 2012. Conceptual modeling of flexible temporal workflows. ACM Trans. Autonom. Adapt. Syst. 7, 2, Article 19 (July 2012), 29 pages.
DOI = 10.1145/2240166.2240169 http://doi.acm.org/10.1145/2240166.2240169

## 1. INTRODUCTION

A workflow schema (*process model*) is a formal description of a business process where single atomic work units (*task)* are assigned to processing entities (*agent*). Instances of a schema are called *cases*. An agent may be a software application (e.g., a database system), a human (e.g., a customer representative), or a combination of both (e.g., a human using a software program). A Workflow Management System (*WfMS*) fully takes over the responsibility for the coordinated execution of tasks and cases: the assignment of tasks to agents is accomplished by enforcing procedural constraints. Organizations use WfMSs to streamline, automate, and manage business processes that depend on information systems and human resources (e.g., provisioning telephone services, processing insurance claims, handling bank loan applications, and managing healthcare processes) [WfMC 1995; Object Management Group 2007].

Adaptivity and flexibility have emerged as strong requirements for WfMSs: indeed, the complexity and the fast evolution and change of business processes and of the related organizational needs should be suitably supported by *adaptive* WfMSs, by

Authors' addresses: C. Combi and R. Posenato (corresponding author), University of Verona, Italy; email: roberto.posenato@univr.it; G. Pozzi, Politecnico di Milano, Italy.

DOI 10.1145/2240166.2240169 http://doi.acm.org/10.1145/2240166.2240169

allowing to: (i) manage exceptions during a workflow execution [van Hee et al. 2007], (ii) change the process model according to some new organizational needs [Nurcan 2008], and (iii) manage different temporal aspects of workflow as temporal constraints and deadlines both at design time and at runtime [Eder and Panagos 2000; Combi and Pozzi 2004].

Even though all these needs are not completely orthogonal, in this article we will specifically focus on the management of temporal aspects by the *WfMS*. Research on WfMSs started to consider the management of temporal aspects in the last decade [Combi and Pozzi 2003; 2004; Marjanovic and Orlowska 1999; Eder et al. 1999; Chen and Yang 2007]: one of the most critical needs in companies striving to become more and more competitive is the ability to control in a flexible and adaptive way the flow of information and of work throughout the enterprise in a timely manner [Eder and Panagos 2000]. Especially in this context, time management is important to timely schedule the workflow process execution, to avoid deadline violations, and to improve the workflow turnaround times. Many business processes have restrictions such as a limited duration of subprocesses, terms of delivery, dates of resubmission, or activity deadlines. Generally, time violations increase the cost of a business process because they lead to some form of exception handling [Eder et al. 1999].

By *temporal flexibility*, we mean that the adaptive *WfMS* is able to represent and manage both at design and at runtime workflows containing temporal constraints among tasks [Rinderle et al. 2004]. At design time, temporal flexibility means that the *WfMS* is able to verify that a workflow schema can be suitably executed according to the given temporal constraints. At runtime, temporal flexibility means that the *WfMS* is able to reassign task durations according to the previously executed tasks and to their durations, and to check whether it is possible to end in a proper way the workflow execution. With regard to temporal flexibility, adaptive WfMSs are required: indeed, a process could need to be changed as its tasks have to be executed in a shorter time than the specified one at design time. Moreover, effective exception handling mechanisms are needed because it could be that some process execution fails to verify all the given temporal constraints, but the execution has to be suitably managed to reach an acceptable conclusion (as, for example, in healthcare processes). According to this scenario, in this article we propose a new framework for the conceptual design of workflows, which takes into account the modeling and the management of temporal aspects in WfMSs. In particular, we focus on the following specific features.

—*Temporal conceptual modeling of flexible workflows.* We present an advanced workflow conceptual model close to the WfMC Reference Model [WfMC 1995], capturing the main temporal aspects of workflow activities (e.g., minimum and maximum durations, delays, temporal relations between activities, deadlines, and other temporal constraints). In particular, we allow the designer to explicitly model temporal constraints that the *WfMS* has to consider when it has to be temporally adaptive with respect to the duration of task executions.
—*Checking the consistency of temporal workflow schemata.* We propose a general method to determine if, given a workflow schema based on our conceptual model, there can exist workflow executions (possibly involving different tasks) where temporal constraints are satisfiable. A schema that admits such cases is called consistent. We will show that the consistency in isolation of any possible workflow execution is not enough for guaranteeing the consistency of the overall workflow schema. Moreover, we propose different kinds of workflow consistency that can be used by the *WfMS* to manage with flexibility the task executions.
—*Supporting run-time flexibility through* WfMSs. We analyze the implications of the different kinds of the workflow consistency on the behavior of WfMSs at runtime, that

is, during case executions. The kind of consistency fixes how to set and to manage the allowed duration ranges for tasks at runtime. Moreover, some kinds of consistency allow us to know in advance that the execution of some tasks (identified by the consistency analysis) can require a workflow schema adaptation or an exception handling. Therefore WfMSs can be programmed in more detail about the management of such events with respect to a generic programming of exception handling.

Furthermore, we propose an XML-Schema document which defines all the components of the model and their temporal attributes as an extension of the XML-Schema document proposed by the WfMC in WfMC [2002]. As a proof-of-concept we extended the well-known Yet Another Workflow Language (YAWL) [van der Aalst et al. 2004], to deal, both at design and at runtime, with temporal workflow schemata based on the proposed temporal workflow model. The prototype allows one to translate workflow schema specifications into XML-Schema representations compatible with the WfMC specification.

The article is structured as follows: in Section 2 we present a conceptual model, mainly focusing on the representation of workflow temporal aspects. In Section 3 we propose a general method for checking the temporal consistency of a workflow schema. We introduce different levels of consistency for a workflow schema and discuss how such consistencies can be used to characterize the soundness and flexibility of workflow schemata at design time and the behavior of the WfMSs at runtime. In Section 4 we consider the proposals from the literature having some relevance for workflow design and compare them with the one we propose in this article. In Section 5 we briefly present the XML representation of the constructs of our model and a prototype of an extension of YAWL to support our proposal. Finally, in Section 6 we sketch out some concluding remarks.

## 2. MODELING TEMPORAL WORKFLOWS

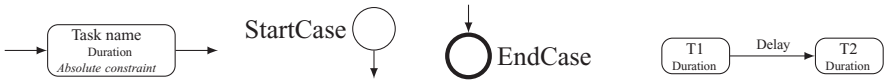
Workflow modeling is an effective technique for understanding, automating, and documenting business processes. Support for heterogeneous processes (human-centered and system-centered), adaptability, flexibility, and reuse are important challenges for the design of process modeling languages [van der Aalst et al. 2002]. As flexible and collaborative processes require human intervention, we need a process modeling language at a high level of abstraction, easy to use, and supporting the visualization of its elements. Additionally, a conceptual workflow model produces high-level specifications of workflows that are independent from the workflow management software.

The conceptual model presented in this article focuses on formalizing all the temporal aspects related to single atomic activities (tasks) and their temporally adaptive activation sequences. By using our model, temporal properties of workflow schemata can be deeply modeled. Our temporal conceptual model is not influenced by any particular commercial *WfMS*, and it is close to the recommendations from the WfMC [WfMC 1995]. The graphical notation we introduce for the constructs of the conceptual model is a straightforward extension of the widely adopted Business Process Modeling Notation (BPMN) [Object Management Group 2007] to consider temporal aspects.

### 2.1. The Temporal Conceptual Model

Conceptual models allow designers to represent workflow schemata (process models) which capture the behavior of processes describing the activities and their execution flow. A workflow schema defines the tasks to be performed, their order of execution, and assignment criteria to agents.

In our model a workflow schema is a directed graph, called *Workflow Graph*. Nodes correspond to activities and edges represent control flows that define the task dependencies that a *WfMS* has to consider when managing the order of execution of tasks.




(a) task representation (b) start and stop symbols (c) a control flow

Fig. 1. Graphic representation of a task (a); Start and Stop symbols (b); and a control flow (c).

*Definition* 2.1 (*Workflow Graph*). A Workflow Graph $WG = (N, T, T_S, T_E, C, F)$ is a six-tuple such that $N = T \cup C$ is a finite set of nodes, $T \subseteq N$ is a finite set of tasks, $T_S \in T$ is the Start node, $T_E \in T$ is the End node, $C \subset N$ is a finite set of connectors, and $F \subset N \times N$ is the control flow relation, that is, the set of edges.

$N$ is the set of all the activities that belong to the process. There are two different activity types: *task* and *connector*. Tasks represent the elementary work units that collectively achieve the process goal. They can symbolize either automated or manual activities that are performed by assigned agents. A task can be initial (Start), final (End), or intermediate. Connectors are the elementary work units executed by the *WfMS* to achieve a correct and ordered sequence for task execution.

The control flow relation $F$ defines the order of execution between any pair of nodes. A path between two nodes represents an order of execution among the set of nodes of the path and it is called *flow*. Given $F$, for each node let us define as *predecessors set* and *successors set* of the node the following sets: (i) the predecessors set of a node $x \in N$ is $\star x = \{y \in N \mid (y, x) \in F\}$, (ii) the successors set of a node $x \in N$ is $x\star = \{y \in N \mid (x, y) \in F\}$.

In the following, we present the syntactic and semantic properties of all the workflow graph components. We restrict ourselves to the main constructs; several other components could be straightforwardly added, for example, supertasks and multitasks [Object Management Group 2007].

*2.1.1. Tasks.* A task is the basic modeling object in workflow schemata and represents the atomic unit of work to be executed. The aim of workflow modeling is to capture the coordination requirements to execute a set of tasks for a given business process. All the other modeling objects but tasks are internal to the *WfMS* and are used to specify rules and constraints for the adaptive coordination of workflow execution. Tasks have many properties, represented by attributes. Task *name* and execution *duration* are mandatory attributes, while *absolute constraint* is an optional one. Duration specifies the allowed temporal span of the activity, while absolute constraint represents a timestamped interval the task must be executed within. More details on temporal aspects are in Section 2.2.2. Every task has one incoming edge and one outgoing edge (see Figure 1(a)).

*2.1.2. Connectors.* Connectors represent internal activities executed by the *WfMS* to achieve a correct and coordinated execution of tasks. Differently from tasks, connectors are directly executed by the *WfMS* and do not need to be assigned to any agent; the mandatory duration attribute of a connector specifies the temporal span allowed to the *WfMS* for executing the connector activity. The graphic representation for connectors in a workflow graph (diamond) is different from the graphic representation for tasks (rectangle with rounded corners) to underline the conceptual difference of the corresponding activities.

As in the WfMC Reference Model [WfMC 1995], in our model we have two connector types: split and join. As depicted in Figure 2, split connectors are nodes with one incoming edge and two or more outgoing edges: split connectors are routing connectors specifying that, after the execution of the predecessor, several successors have to be

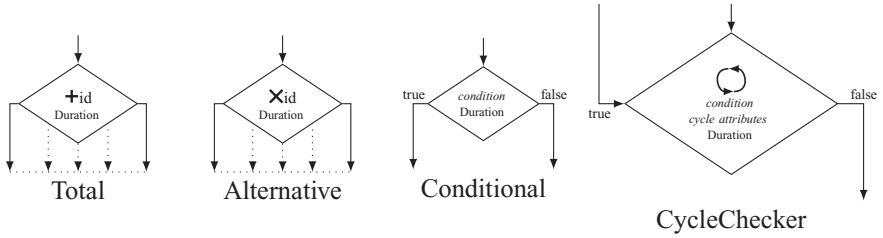



Fig. 2. Graphic representation of split connectors as proposed by Object Management Group [2007].

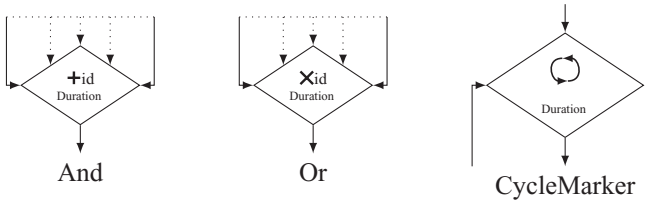



Fig. 3. Graphic representation of *join* connectors as proposed by Object Management Group [2007].

considered for the execution. The set of nodes that can start their execution is given by the features of each split connector. A split connector can be one of the following.

—*Total.* It specifies that all successors are triggered for a parallel execution. Each successor execution is independent from the others, so the incoming flow splits into as many independent flows as the outgoing edges are.
—*Alternative.* It stands for a choice node. It has two or more outgoing edges representing mutually exclusive (alternative) flows. This ensures that only one successor node is selected at runtime. The choice of the successor node depends on the evaluation of some system conditions such as the availability of agents or some other external conditions.
—*Conditional.* It is a specialization of the *Alternative* connector where there are only two alternative successors and the choice depends on the state of some workflow variables instead of some system or external conditions. The boolean expression that represents the choice is called *condition*. After the evaluation of the condition, the successor task associated to the corresponding true value is triggered for the execution.
—*CycleChecker.* It represents the control of a cycle and it has the same properties of the conditional connector. Additionally, it defines suitable cycle attributes: *iterations* and *deadline*. Iterations are represented as an interval expressed as the minimum and the maximum number of allowed iterations. Deadline is the maximum allowed duration for completing all the iterations of the cycle. At least one of these two attributes must be set by the designer to ensure the cycle termination.

Join connectors are nodes with two or more incoming edges and one outgoing edge only, as shown in Figure 3: join connectors merge more flows into one single flow. A join connector can be one of the following kinds.

—*And.* It joins two or more *concurrent* flows into one flow. It plays the role of synchronizer as it is ready for execution only after the end of the execution of all its predecessors.
—*Or.* It joins two or more *alternative* flows into one. Since alternative flows come from an *Alternative* connector or a *Conditional* one, only one of them can be operative at runtime. Therefore, the connector is ready for execution after the end of the execution of the predecessor present on the running flow.

—*CycleMarker*. It represents the starting point of a cycle. Each cycle in this model contains a CycleMarker and a CycleChecker connector. The position of activities with respect to these two connectors determines the kind of cycle as explained in Section 2.1.5. The CycleMarker has always two incoming edges: one comes from the predecessor node while the other one comes from the last node of the cycle.

*2.1.3. Start and End.* Figure 1(b) depicts the graphic representation for Start and End symbols. These constructs enable the creation and the completion of workflow cases and do not represent proper activities: they have no temporal property. In detail, *Start* represents the start symbol of a workflow graph. After the creation of a workflow instance, the Start successor is triggered for execution. Each workflow graph must have exactly one Start. *End* represents the stop symbol of a workflow graph. Each workflow graph must have one End. When End is reached, the workflow is completed.

*2.1.4. Control Flow.* Recalling that the control flow relation $F$ defines the order of execution between any pair of nodes, in Figure 1(c) there is an example of a control flow between two tasks. The edge orientation suggests the direction of the control flow: the predecessor must be finished before starting the execution of the successor. Every edge has a temporal property, *delay*, that denotes the time that can be used by the *WfMS* for its internal activities between the end of the predecessor and the beginning of the successor and for managing in an adaptive way the execution of workflow cases according to the given temporal constraints. More details are given in Section 2.2.

*2.1.5. Cycles.* The cycle construct allows one to model the iteration of a group of activities within a workflow schema. To efficiently manage cycle constructs, some rules are needed. Every cycle is a subgraph in the workflow schema with one CycleMarker as starting node and one CycleChecker as ending node of the subgraph, respectively. The activities of a cycle can be positioned between CycleMarker and CycleChecker or between CycleChecker and CycleMarker, depending on the kind of cycle wanted.

All the activities belonging to the cycle are repeated as long as the CycleChecker conditions hold. The workflow execution leaves a cycle if any of the following conditions holds: (i) the CycleChecker condition becomes false; (ii) the number of iterations reaches the maximum number of allowed iterations specified in CycleMarker; (iii) the cumulative time of the executed iterations exceeds the deadline value specified in CycleChecker.

We observe that if the iterations attribute is specified, the workflow can be rewritten into a cycle-free one. In fact, the cycle subgraph can be replaced by a subgraph containing iterations copies of the original cycle linearly connected as follows: all CycleMarker connectors are removed and each CycleChecker one becomes a Conditional connector where the true-value edge connected to the first node of the following copy and with the false-value edge connected to the first node after the last copy; the condition of these Conditional connectors is the original condition of the CycleChecker.

*2.1.6. Well-Structured Workflow Graphs.* In the following we assume workflow schemata containing split connectors are *full-blocked* schemata as defined by WfMC [WfMC 2002], where each split node has a corresponding join node and vice versa. Moreover, cycles are properly nested. For more details about syntactical correctness issues and requirements, please see online Appendix A.

*2.1.7. Workflow Paths (wf-paths).* Due to conditional and alternative flows, not all the cases of one workflow schema perform exactly the same set of activities. We group workflow cases into *workflow paths* (*wf-paths*) in accordance with the activities actually executed. A *wf-path* can be regarded as a workflow subgraph in which every alternative or conditional connectors has exactly one successor [Eder and Gruber 2002]. When a

schema contains a cycle, there are as many *wf-paths* as the possible iterations. It is sufficient to unroll the cycle as shown in Section 2.1.5: each new Conditional connector determines a new *wf-path*. Since in our conceptual model any task cannot have a 0 duration time (see Section 2.2 for the motivation), the number of iterations of a cycle cannot be infinite even if the designer sets only the deadline of the cycle. Thus, the number of *wf-paths* that represent all the cycle iterations is finite and, in the worst case, it is exponential with respect to the dimension of the workflow encoding.

In order to shortly denote a *wf-path*, hereinafter we will use the string built by means of the name of tasks (connectors are ignored) present on *wf-path* separated by an "-" if two tasks are in sequence or by " || " if they are on parallel flows (in this case, parallel tasks are written according to the lexicographical order of their names). For example, in Figure 4, *wf-paths* (b) is denoted by T1||(T3-T4) while *wf-paths* (c) is denoted by T1||T2. If a workflow schema has no alternative or conditional connectors, one *wf-path* only occurs.

### 2.2. Modeling Time and Temporal Aspects

In this article, we describe how time constraints (i.e., upper and lower bound of tasks durations, minimum and maximum delays between activity executions, fixed-date constraints) can be captured during the conceptual process definition. Therefore, the basic temporal concepts used in workflow processes are introduced and the conceptual modeling of such kind of temporal information is presented.

Our model considers instants and durations as elementary temporal types [Goralwalla et al. 2001]. Instants are points on the time domain, while durations are lengths on the time domain. Intervals are derived types and can be defined as the time span between starting and ending instants. Instants are represented through timestamps: each timestamp is defined at a given granularity. In this article we adopt the approach proposed by Goralwalla et al. [2001] for modeling granularities: a (calendric) *granularity* is a unit of measurement for spans of time. For example, the granularity of days (day) stands for a duration of 24 hours. More generally, a granularity is a special kind of, possibly varying, duration that can be used as a unit of time. Such granularities may thus be used as a unit of measure for expressing durations and also for specifying time points; in such a case, granularities are used for expressing the distance of a time point from a *reference time point*, chosen as origin of the time axis. In the following, without loss of generality we adopt the granularities of the Gregorian calendar, that is, year, month, week, day, hour, min, as units of measure for expressing durations and the International standard date notation (ISO 8601:2004 [ISO 2004]) to represent dates and times (i.e., locations of time points): therefore, for example, a week stands for a duration of 7 days while 2007-10-13 08:00 represents the instant "8 o'clock on 13th, October 2007".

*2.2.1. Task Durations and Delays.* The duration is a temporal property of both nodes and edges. A node duration represents the temporal span of the corresponding activity and is the temporal distance between the starting and ending instants of the activity. The edge duration, called *delay*, denotes the time span between the ending instant of the predecessor and the starting instant of the successor, that is, the duration at disposal of the *WfMS* for managing the start of the successor(s) after the end of the predecessor(s). Even if a delay can be simulated by a dummy task and this substitution could simplify the workflow graph structure, we prefer to maintain the delay concept to underline it represents a time span that could be spent by the *WfMS* to adaptively coordinate the task executions instead of a time span that is available to a dummy agent.

In the real world, workflow designers cannot use precise values for durations of node and edges. For example, tasks can be performed by human agents, and generally

the duration of a task cannot be precisely known at workflow design time; on the other hand, even for internal activities (e.g., a split) it is not possible to precisely know how much the *WfMS* will take to perform them, depending on the workload of the system. Moreover, delays could be set by the *WfMS* to correctly manage the overall temporal constraints the workflow execution has to satisfy. Therefore, allowed durations should be expressed by using ranges like expected duration $\pm$ time tolerance. Our conceptual model describes all the durations by an attribute having the form [MinDuration, MaxDuration] Granularity with $0 \leq$ MinDuration $\leq$ MaxDuration, as depicted in Figure 4. Range bounds represent the minimum and maximum allowed durations. The minimum duration can represent either an estimate of the required time to carry out the activity or a constraint to the necessary time to carry out it. For example, to determine a parameter during a blood analysis it would be necessary to wait until a reagent reacts before proceeding with the measurement. The maximum duration represents a *deadline* to the time to carry out the activity. In our conceptual model, each workflow component has the duration attribute. If the workflow designer does not set a duration, we set the attribute value to [1,$+\infty$] MinGranularity, where MinGranularity is the finest granularity managed by the *WfMS*.

We do not admit [0,0] MinGranularity to underline that no activity can be executed without time consumption. A user can always set [0,n] Granularity as a duration attribute to specify that an activity can last 0 Granularity if Granularity is not the minimal granularity used by the *WfMS* to measure time.

*2.2.2. Temporal Constraints.* Besides the basic temporal constraints, expressed through durations as previously described, our conceptual model allows one to express several kinds of temporal constraints: from the business perspective they are defined by laws and regulations, business policies, common practices, as well as mutual agreements and expectations related to efficiency of business practice [Marjanovic and Orlowska 1999]. Differently from the duration attributes, temporal constraints are not mandatory for each workflow component: they model additional temporal properties and must be controlled by the *WfMS*. We classify temporal constraints for workflows as *relative* constraints and *absolute* constraints.

A relative constraint limits the time distance (duration) between the starting/ending instants of two nonconsecutive workflow nodes. Our model provides one type of relative constraint, expressed according to the notation $I_F$[MinDuration, MaxDuration]$I_S$ Granularity, where: (i) $I_F$ marks which instant of the first node to use ($I_F$ = S for the starting execution instant or $I_F$ = E for the ending one) while $I_S$ marks the instant for the second node; (ii) [MinDuration, MaxDuration] Granularity with MinDuration $\leq$ MaxDuration represents the allowed range for the time distance between the two instants $I_F$ and $I_S$.

A finite positive MaxDuration value models a deadline as defined in other workflow models [Eder et al. 2000; Marjanovic and Orlowska 1999], since it corresponds to the maximum global allowable execution time for the activities that are present on possible flows between the first node and the second one. On the other hand, a finite positive MinDuration represents the minimum execution time that has to be spent before proceeding after $I_S$: if the global time spent to execute all activities between $I_F$ and $I_S$ is less than MinDuration, then the *WfMS* has to dynamically manage a suitable exception (like to sleep, for example) that depends from the specific applications. A finite negative MaxDuration value expresses that the $I_S$ has to occur before $I_F$ |MaxDuration| instants at least.

Figure 4(a) depicts two examples of relative constraints. The first one is the edge connecting Start to T3. It fixes to 18 minutes the maximum time distance and 1 minute

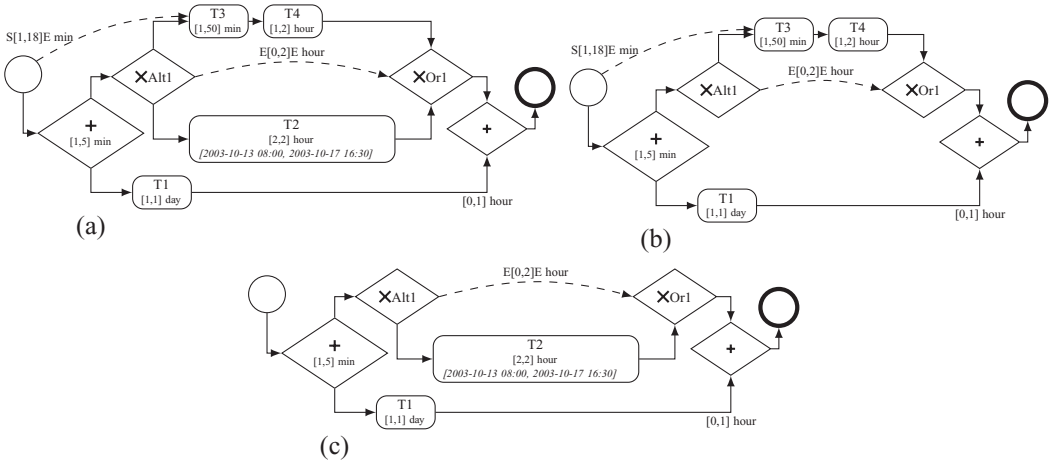



Fig. 4. Workflow graph(a) and its two *wf-paths* (b) and (c). The workflow contains also two relative constraints (represented as dashed edges) and one absolute constraint on task T2.

the minimum one between the workflow execution start and T3 end; this constraint has to be evaluated only in *wf-paths* that contain T3.

Relative constraints are conceptually more expressive than the deadline constructs used in other models. In fact, relative constraints can model other temporal bindings, as depicted by the Alt1-Or1 relative constraint of Figure 4(a). This constraint fixes to 2 hours the maximum time distance between the end of Alt1 and the end of Or1 and it has to be evaluated in every *wf-path* allowing therefore a global constraint on the duration of alternative flows of the workflow schema. Relative constraints cannot be set for activities belonging to mutually exclusive flows. Tasks within a cycle may be constrained only with respect to other tasks of the same cycle and these constraints are verified each iteration of the cycle. The CycleChecker and the CycleMarker nodes can be constrained with respect to any task inside or outside the cycle.

An absolute constraint represents a time interval during which an activity can be performed; interval bounds are expressed by timestamps. Bounds of an absolute constraint are independent from the workflow execution starting point. Figure 4(a) depicts an example of an absolute constraint: task T2 must be completed within the interval from 2003-10-13 08:00 to 2003-10-17 16:30. The span of the time interval for the absolute constraint must be greater than or equal to the maximum duration defined for the corresponding activity.

### 2.3. A Motivating Example of a Temporal Workflow Schema

As an example of a real workflow schema, we consider here an excerpt from the guideline for the diagnosis and treatment of *ST-segment Elevation Myocardial Infarction* (STEMI), published by the American College of Cardiology/American Heart Association in 2004 [Antman et al. 2004]. Figure 5 depicts the corresponding temporal workflow schema. The case starts as the patient is admitted to the Emergency Department (ED) (task T1) that has not to require more than ten minutes. After the admission, the patient is examined by a physician (task T2) that can take a time between five and twenty minutes to make the examination. If the diagnosis is a Class-I STEMI occurrence (connector C1), then a well-known set of therapy and diagnosis activities has to fire (*true* flow). Otherwise, a further patient evaluation has to be done (*false* flow). Since the guideline mainly considers Class-I STEMI patients, we have decided to close

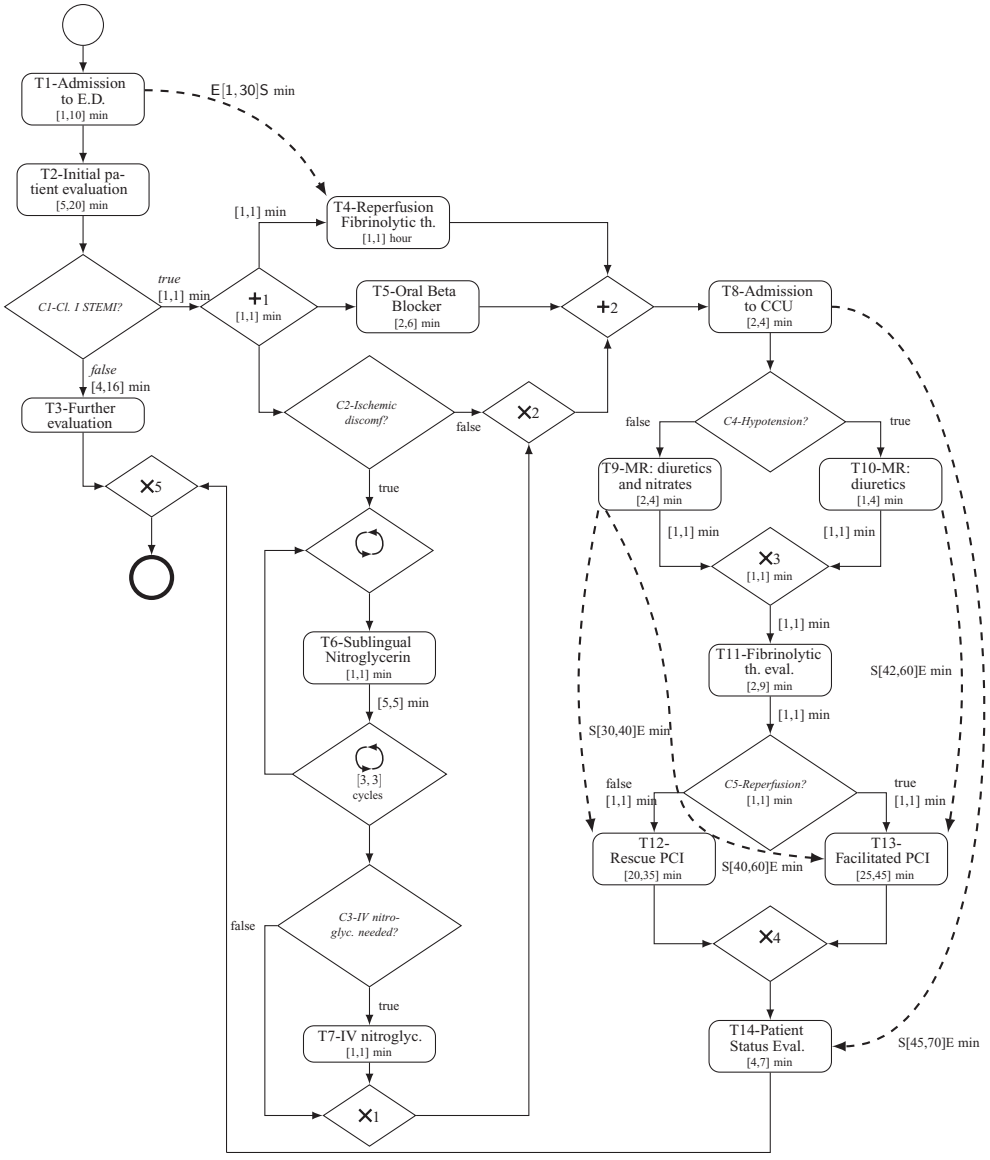



Fig. 5. Workflow graph example of patient admission to an hospital. The dashed edges represent relative constraints. It is worth noting that the last Or-join ×5 could be avoided, maintaining the same expressivity, if we allow more Ends, one for each alternative path.

the false flow by a generic task (task T3) to represent a further evaluation. The true flow is composed by three parallel flows starting from connector +1.

The lowest flow contains the (possible) activities related to therapies for ischemic discomfort: after the evaluation of the presence of ischemic discomfort (conditional connector C2), three administrations of sublingual nitroglycerin are performed (cycle with internal task T6); a successive intravenous nitroglycerin is administered (connector C3 and task T7), as needed. The central flow refers to the complementary therapeutic action consisting of the assumption of beta blocker drugs (task T5). The uppermost flow refers to the main therapeutic action in presence of a myocardial infarction: reperfusion

is obtained through a fibrinolytic therapy (task T4). After all these therapeutic actions, the following activity (task T8) consists of admitting the patient to the Coronary Care Unit (CCU). After the admission to CCU (lasting at most four minutes), a check about the blood pressure of the patient is performed and, according to this check, two alternative therapies are provided: in case of hypotension, the physician has to review the medication (Medication Review (MR)) about diuretics (task T10) while, in case of absence of hypotension, the physician has to review the medication about diuretics and nitrates. Afterwards, an evaluation of the fibrinolytic therapy is needed (task T11). According to the result of this evaluation, two alternative Percutaneous Coronary Intervention (PCI) are executed: indeed, if the fibrinolytic therapy did not obtain reperfusion, a rescue PCI is promptly performed (task T12) while if the patient is reperfused, a facilitated PCI is allowed (task T13). At the end of the chosen PCI, an evaluation of the overall patient status is done (task T14).

It is possible to observe that there are different temporal constraints for tasks, given at different granularities. The intertask constraint between the end of task T1 and the beginning of task T4 represents the most important recommendation from the guideline to successfully apply the fibrinolytic therapy to patients. Moreover, temporal constraints between tasks T9 and T12, T9 and T13, T10 and T13 limit the time span between the definition of therapies and the execution of PCIs. A further constraint between T8 and T14 specifies a possible range of durations for the activities in the CCU.

If durations are not specified, then they are set to the default value of $[1,+\infty]$ min since we assume that the minimum granularity managed by the *WfMS* is minute.

After the definition of the STEMI workflow schema, it is important to verify if it is possible to carry out a successful execution: for example, an obvious question we need to answer is whether it is possible to have the fibrinolytic therapy (task T4) within the allowed time span with respect to the admission of the patient to the Emergency Department (task T1) when the activities and delays between them assume allowed time durations. Another possible question is whether the final evaluation of the patient status (task T14) may be executed satisfying the temporal constraint with respect to the admission of the patient to the CCU (task T8) disregarding the fact that the patient undergoes a rescue PCI (task T12) or a facilitated one (task T13). All these kinds of verifications may be termed *design-time evaluation*.

After the design-time evaluation, a further evaluation has to be assessed at runtime (*runtime evaluation*): for example, if the admission to the ED (task T1) is performed in 8 minutes and the following task T2 is started one minute after, we need to reevaluate the allowed durations for task T2 in order to be able to start the fibrinolytic therapy (task T4) within 30 minutes after the admission to ED. Moreover, we could need to reevaluate the allowed durations for the patient status evaluation (task T14) when the specific sequence of performed tasks does not involve some temporal constraints (e.g., for the given patient task T9 and task T12 have been executed and therefore the constraint between T9 and T13 is not meaningful).

## 3. CHECKING TEMPORAL WORKFLOW SCHEMATA

The successful completion of a process often depends on the correctness of temporal aspects modeled at design time. If temporal constraints are such that any of them cannot be satisfied, the process cannot be performed successfully. Therefore, preliminary temporal evaluations are needed to state whether the specified temporal constraints can be satisfied by any workflow instance, where satisfaction means the existence of at least one assignment of timestamps to the start and to the end instants of activities such that all constraints hold. When one of such assignments exists, the workflow schema is called *consistent*. These preliminary temporal evaluations can be complex because a

workflow schema may represent many *wf-paths* and therefore the evaluations have to be done for each flow separately and suitably merged at the end.

In this section we propose a general method to check the consistency of temporal schemata and, since there are some different levels of constraint satisfaction, we propose a classification of workflow schemata in accordance with different kinds of temporal workflow consistency. This classification can be useful to evaluate, for example, the quality of a schema and its flexibility both at design time and at runtime. More particularly, first we distinguish the design-time evaluation from runtime evaluation of temporal workflows constraints and, then, we discuss separately each of them.

### 3.1. Evaluating Temporal Constraints for Workflows

For brevity's sake, henceforth we name without distinction a duration attribute or an absolute/relative constraint or a delay as a *temporal constraint*.

In general, the analysis of temporal constraints of a workflow is a three-stage process that starts just after the schema design completion and ends when an instance execution ends.

(1) *Design-time evaluation.* It is executed only once, just after the workflow schema design completion. The goal is to verify all temporal constraints but absolute ones. It consists of:
   (a) converting all the temporal constraints into equivalent ones at the finest granularity;
   (b) verifying the satisfiability of all the temporal constraints but absolute ones.

   Absolute constraint satisfiability cannot be verified because in this stage only the workflow schema is analyzed and, therefore, there is not a starting execution time necessary to consider temporal absolute constraints.

   The result of this stage is a successful validation or a rejection of the schema when at least one nonsatisfiable constraint is found. If the schema is validated, all its temporal constraints but absolute ones are expressed in terms of the same temporal granularity.

(2) *Runtime evaluation.* It runs in parallel with the workflow instance execution and its goal is to continuously update the active temporal constraint values given the actual times of completed activities.

   This stage can be divided into two different substages.
   (a) *Start-time evaluation.* It starts when a starting time for a workflow instance is given. Its goal is to convert and to verify all absolute temporal constraints. These constraints are converted into relative equivalent ones using the workflow instance starting time. Afterwards, all the constraints (new ones included) are verified again in the same way as done in the first stage. If all the constraints are satisfiable, the schema instance is validated and its execution can start at the given starting time.
   (b) *Node-by-node evaluation.* As an activity completes, if it is involved in some relative constraints, its actual execution time can compel changes for other activities/delays involved in the same relative constraints. These changes can require the *WfMS* to reduce the allowed duration ranges of some subsequent activities or to increase the delay before or between two subsequent activities in order to guarantee the overall constraint satisfaction. On the other hand, some constraints could be relaxed at runtime because some *wf-paths* are no longer possible due to previously executed alternative/conditional splits.

   For example, let's consider a simple workflow where there are three sequential tasks. Each task has to be completed in 6 minutes at most and there is a relative constraint stating that all tasks have to be completed in 10 minutes at most. The

constraints are satisfiable. If the first task lasts 6 minutes, the second and third ones cannot last more than 4 minutes globally. Therefore, after the first task completion, the *WfMS* has to determine new ranges of allowed durations for the following tasks, before assigning them to agents.

A general method to check constraint consistency in these two stages can be based on the general method for the Simple Temporal Problem (STP) consistency check [Dechter et al. 1991].

In the following sections we describe the two stages in more details. First of all, we show how to convert possible different temporal granularities into a common one and how to convert absolute constraints into relative equivalent ones once the workflow instance starting time is given. Then, we recall some concepts on STP and on its consistency check; moreover, we show the equivalence between a *wf-path* and an STP and, therefore, that the consistency check of a *wf-path* is equivalent to check the consistency of the corresponding STP. We define the workflow consistency in accordance with consistencies of its *wf-paths* and we also propose three types of temporal consistency of a workflow schema according to the features of its *wf-paths*.

Finally, we discuss how these three types of consistency have effect on the behavior of a workflow execution and how they can require different degrees of temporal adaptiveness to the *WfMS*.

### 3.2. Conversion of Temporal Constraints

To verify temporal constraint consistency, we need to convert all the temporal constraints specified in the workflow schema at different granularities into the equivalent ones at the finest granularity managed by the *WfMS*. Absolute constraints have to be previously converted into relative ones.

Temporal constraints are always interpreted as allowed time spans between time points at the finest granularity [Goralwalla et al. 2001]. For instance, assuming that min is the finest granularity managed by the *WfMS*, the constraint [0,0] day will be translated into the equivalent constraint $[1, 24 \cdot 60 - 1]$ min, as the lower bound of the range is the minimum duration for a span lasting 0 day and the upper bound $24 \cdot 60 - 1$ is the maximum duration for a span lasting less that 1 day, that is, 24 hours.

To deal with the temporal properties that depend on the starting point of the workflow instance, it is necessary to convert the fixed-date bounds of absolute constraints into bounds that are dependent from the starting instant of that workflow instance: this conversion is obtained by subtracting the starting instant of the workflow instance from the interval fixed-date bounds. As regards the workflow graph, an absolute constraint conversion is represented by a pair of relative constraints: one between the Start and the start instant of the node and the other one between the Start and the end instant of the node. Both constraints have the same converted temporal range, mentioned before.

### 3.3. Temporal Workflows and the Simple Temporal Problem

Before explaining how we check the consistency of a workflow, it is useful to demonstrate that the consistency of *wf-paths* defined by our conceptual model can be checked by using polynomial-time algorithms in similar way as done by Dechter et al. for Simple Temporal Problems (STPs) [Dechter et al. 1991].

For sake of simplicity, we recall that we want to check whether all temporal constraints of a *wf-path* are consistent, where consistency means the existence of at least one assignment of timestamps to start and end times of activities, such that all constraints hold. The constraints that may cause the overall consistency to fail are the relative ones: without them, consistency is guaranteed as it is always possible to find an assignment to start and end times of activities that satisfy the single specific

constraint on activity durations and delays. Instead, when relative constraints are set, it is possible that a relative constraint on the time distance between activities $A$ and $B$ cannot be satisfied, since the sum of the minimal durations for the activities and edges from $A$ to $B$ is larger than the maximum duration the constraint allows. Similarly, the sum of the maximum durations of the activities and edges may be smaller than the minimum time distance in the constraint. Moreover, even if a path is consistent, it is possible that for an activity with constraint $[a, b]$, all satisfying assignments fall into $[a', b']$, where $a < a'$ and $b' < b$. The maximum is reduced since a relative constraint applies, and the sum of the minimal durations for other activities does not allow this activity to execute in more than $b'$. A symmetric argument holds for the minimum. In this case, we would like to find $a'$ and $b'$. This analysis (and eventually new intervals determination) can be carried out by showing that a *wf-path* is equivalent to an instance of the STP and using the well-known results stated for it. An STP consists of a set of variables, $\{X_1, \ldots, X_n\}$, specifying time points, and a set of constraints for the distances between these variables. Each constraint is specified as a single range of allowed values between two variables and there cannot exist more than one constraint in the set for a given pair of variables. Any constraint is in the form $a \leq X_j - X_i \leq b$. A solution is an assignment $\{X_1 = v_1, X_2 = v_2, \ldots, X_n = v_n\}$ such that all constraints are satisfied. An STP can be associated to a directed weighted graph, called *distance graph*. Its nodes are the variables and its edges are the constraints. For each constraint $a \leq X_j - X_i \leq b$, two edges $(X_j, X_i)$ and $(X_i, X_j)$ are defined with weights $-a$ and $b$ respectively. An *STP* is *consistent*, that is, all its constraints can be satisfied, if its distance graph has no negative cycles. An algorithm to detect negative cycles is the Floyd-Warshall's all-pairs shortest-path algorithm [Cormen et al. 2001]. Starting from a distance graph, this algorithm yields a complete weighted graph, called *d-graph*. If the *d-graph* has all positive self-loops then the distance graph has no negative cycles and each $(i, j)$ edge label is the shortest path length between $i$ and $j$ in the distance graph. Each STP constraint range obtained from these shortest path lengths is a subrange of the original one [Dechter et al. 1991]. The Floyd-Warshall's algorithm has complexity $O(n^3)$, where $n$ is the number of nodes [Cormen et al. 2001]: hence the problem of consistency for STPs can be solved in time $O(n^3)$. Here we shall prove that every *wf-path* of our model can be expressed as an STP.

THEOREM 3.1. *The temporal wf-paths of our conceptual model represent simple temporal problems.*

PROOF. To demonstrate the theorem we only need to show that a *wf-path* can be translated into an equivalent STP.

The translation has to transform the activity nodes and edges with their temporal properties into equivalent time point variables and temporal constraints of an STP. This task is accomplished by three steps.

(1) *Nodes conversion.* Given a workflow graph, each node $A$ is converted to two variables, $A_S$ and $A_E$, representing its start and end instants. The duration attribute of $A$, $[a, b]$ is converted to the constraint $a \leq A_E - A_S \leq b$.
(2) *Edges conversion.* An edge from node $A$ to node $B$ with delay $[c, d]$ is converted to the constraint $c \leq B_S - A_E \leq d$.
(3) *Relative constraint conversion.* A relative constraint edge between two nodes $A$ and $B$ is converted to the constraint using the variables that represents the time instants declared in the constraint; for example, the constraint E[1, 30]S min between tasks T1 and T4 in Figure 5 is converted to the constraint $1 \leq \mathsf{T4}_S - \mathsf{T1}_E \leq 30$. □

COROLLARY 3.2. *The consistency of a wf-path graph can be checked in time $O(n^3)$.*

PROOF. As stated in Theorem 3.1, each *wf-path* can be transformed into an equivalent STP problem. Applying the Floyd-Warshall's algorithm to the STP associated distance graph, it is possible to say whether the original problem is consistent, as described at the start of this section. □

### 3.4. Design-Time Temporal Workflow Consistency

Given a workflow schema, its temporal consistency check can be done starting from the consistency check of its *wf-paths*. As shown in the previous section, given a *wf-path*, it is possible to state whether all of its temporal constraints are satisfiable and, if so, to determine the optimal ranges for them in a polynomial time. Optimal range means that for each of its instants there exists at least one temporal solution (i.e., a time assignment to all activities) that uses such duration. Considering all the *wf-paths*, we could say that a workflow schema is *temporally consistent* if all of its *wf-paths* are consistent.

However, from the point of view of the *WfMS*, this näive definition of temporal consistency of a workflow schema does not suffice to successfully manage the tasks and their assignment to agents. For example, it could be that for a task there are disjoint ranges of allowed durations from different *wf-paths*: in this case the actual duration of the task will exclude the possibility of executing some *wf-paths* and this could be a problem since such excluded *wf-paths* could be the required ones to successfully complete the workflow. Another possible situation is when the range for an activity allows the execution of any *wf-paths*. A more subtle situation is when different duration ranges for an activity allow any successive execution flow, but the range is known only when we fix the previously executed activities (i.e., the flow from the start case to the considered activity): in this case the WfMS has to adaptively manage the duration of the considered activity according to the actual execution flow. Therefore, it is necessary to further analyze and detail the concept of workflow consistency to manage the allowed duration ranges in a right way. To do that, we have first to introduce the preliminary concept of *prefix*, to group together *wf-paths* having some common set of activities and edges.

*Definition* 3.3 (*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 $y$ either it belongs to the predecessor set of $y$ or precedes an activity of this set.

Hereinafter a prefix of a given activity/edge $y$ can be represented by the *wf-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, in Figure 5, considering prefixes T1-T2 and T1-T2-(T4||T5), it holds T1-T2 $\preceq$ T1-T2-(T4||T5). It is worth noting that for any prefixes $\alpha, \beta$ such that $\alpha \preceq \beta$ the *wf-paths* of $\beta$ belong to $\alpha$ too, that is, $\alpha \supseteq \beta$.

Considering a given prefix, let's determine for each activity/edge of the prefix the intersection of the allowed ranges in its different *wf-paths*. If all such intersections are not empty, we consider them as new activity/edge ranges and, therefore, we determine the consistency for each *wf-path* of the prefix. If all *wf-paths* are still consistent, it can be that such consistency checks reduced the allowed ranges for some activities/edges: in this case, we reapply the previous steps by deriving the new intersection for all activity/edge ranges and checking again the consistency of each (updated) *wf-path*. These steps have to be iterated until there are no more range modifications. After this iteration, the new ranges represent the duration ranges for activities/edges allowed for any *wf-path* of the given prefix and, therefore, we say that the prefix is *consistent*. In case that any intersection results to be empty or any *wf-path* results to be not

```
Function buildCommonRanges(S)
  Input: S: set of consistent wf-paths
  Output: the check status and a new set of temporal ranges for activities and delays
  do
      foreach task t present into any wf-path do
          t range = intersection of t-ranges present into wf-paths;
      foreach connector c present into any wf-path do
          c range = intersection of c-ranges present into wf-paths;
      foreach delay d present into any wf-path do
          d range = intersection of d-ranges into wf-paths;
      R = collection of these new ranges;
      if (at least one range is empty) then
          return ('emptyRangeFound', ∅)
      foreach wf-path p ∈ S do
          Execute Floyd-Warshall on the distance graph corresponding to p with ranges in R;
  while (at least one range has changed ∧ no range is empty ∧ no wf-path is inconsistent);
  if (at least one wf-path is inconsistent) then
      return ('noConsistentPathFound',∅)
  else if (at least one range is empty) then
      return ('emptyRangeFound', ∅)
  else
      return ('commonRangesFound', R)
```

Fig. 6. A simple algorithm for prefix consistency check. Given a set of *wf-paths* that form a prefix, the algorithm returns the set of activity/edge temporal ranges (if any) that works in any *wf-path*; if any inconsistency arises during the computation of temporal ranges, it returns the empty set. Moreover, it returns a flag that qualifies the set returned.

consistent, it means that for at least one activity/edge there does not exist a common duration range for all *wf-paths* of the prefix and, thus, the prefix is not consistent. Figure 6 presents the algorithm that checks the consistency of a given prefix according to this approach.

As a consequence of this new characterization, it is possible to improve the workflow schema consistency concept by defining three different levels of consistency. To discuss these levels of consistency, we shall consider as a complete workflow the last tasks of the STEMI workflow schema (see Figure 5) starting from task T8 as depicted in Figures 7, 8, and 9: this part of the workflow schema consists of the task T8, followed by either T9 or T10, T11, either T12 or T13, and finally T14; the missing values for delays and durations are set to [1,+∞] min, while all durations for tasks and relative constraints, edges, and connectors are already expressed at the finest granularity of minutes. As there are two alternative splits, we have four different *wf-paths*: T8-T9-T11-T12-T14, T8-T9-T11-T13-T14, T8-T10-T11-T12-T14, and T8-T10-T11-T13-T14, respectively.

The first kind of consistency we focus on is the most desirable one from the planning point of view.

*Definition* 3.4 (*Workflow Strong Consistency*). A workflow schema is *strongly consistent* if the prefix of Start is consistent.

If a workflow is strongly consistent, a first consequence is that the *WfMS* can communicate a preliminary task duration range to every agent. Moreover, the time required by an activity does not affect the following possible flows, but only the duration range for the following activities. It is worth noting that at runtime, as we will discuss in the

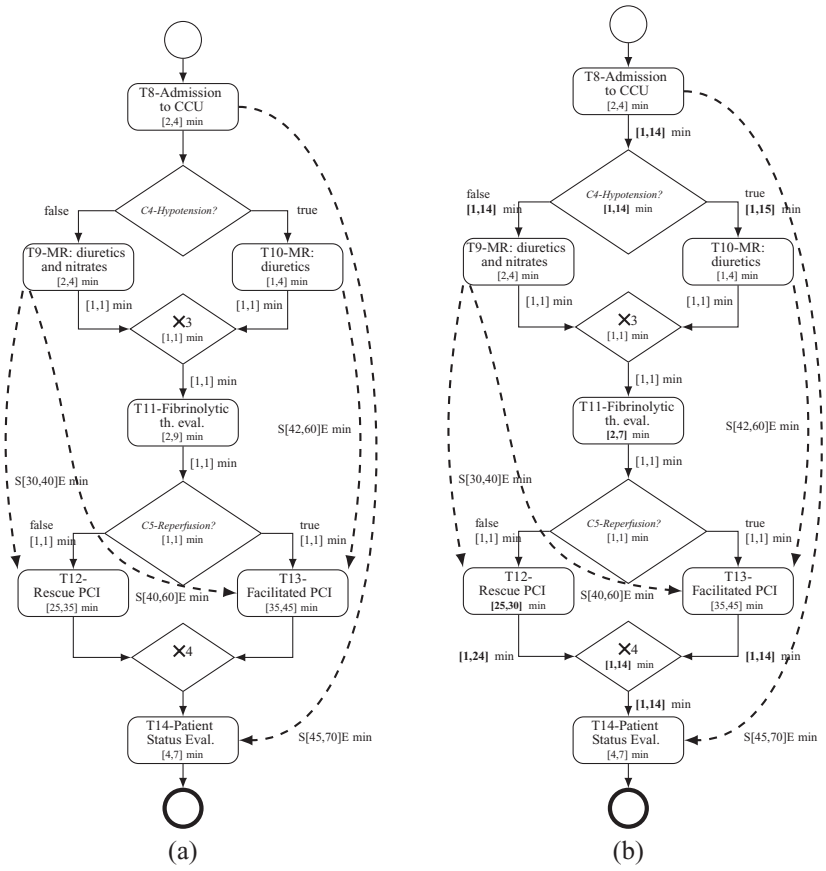



Fig. 7. Strong consistency case: (a) Original workflow graph schema; (b) the result of consistency analysis. The schema is strongly consistent; ranges for activities/edges are the intersections of all *wf-paths* ranges calculated. The boldface ranges are different from the corresponding ones in (a).

next section, this range could be adaptively enlarged or restricted in accordance with the execution time of the previous completed tasks. An example of a strongly consistent workflow schema is given in Figure 7(a). All *wf-paths* are consistent in isolation. The prefix of Start (i.e., the set of all four *wf-paths*) results to be consistent and the new derived ranges are depicted in Figure 7(b). For example, task T11 has the reduced range [2,7] min.

The second type of consistency is the history-dependent one.

*Definition* 3.5 (*Workflow History-Dependent Consistency*). A workflow schema is *history-dependent consistent* if: (i) the prefix of Start is not consistent, and (ii) there exists at least one ✕ join connector such that all its prefixes globally contain all the possible *wf-paths* and each of its prefixes is consistent. The consistency of prefixes is evaluated in the following way: the allowed range of each activity/edge preceding the given ✕ join is restricted to have a single value for each prefix of the considered activity/edge.

If a workflow is history-dependent consistent, at least one activity has more than one disjoint duration range. Each duration range is associated to one or more prefixes of the activity (this explains the expression "history-dependent") but duration ranges do not restrict any possible following flow. The *WfMS* can communicate a preliminary

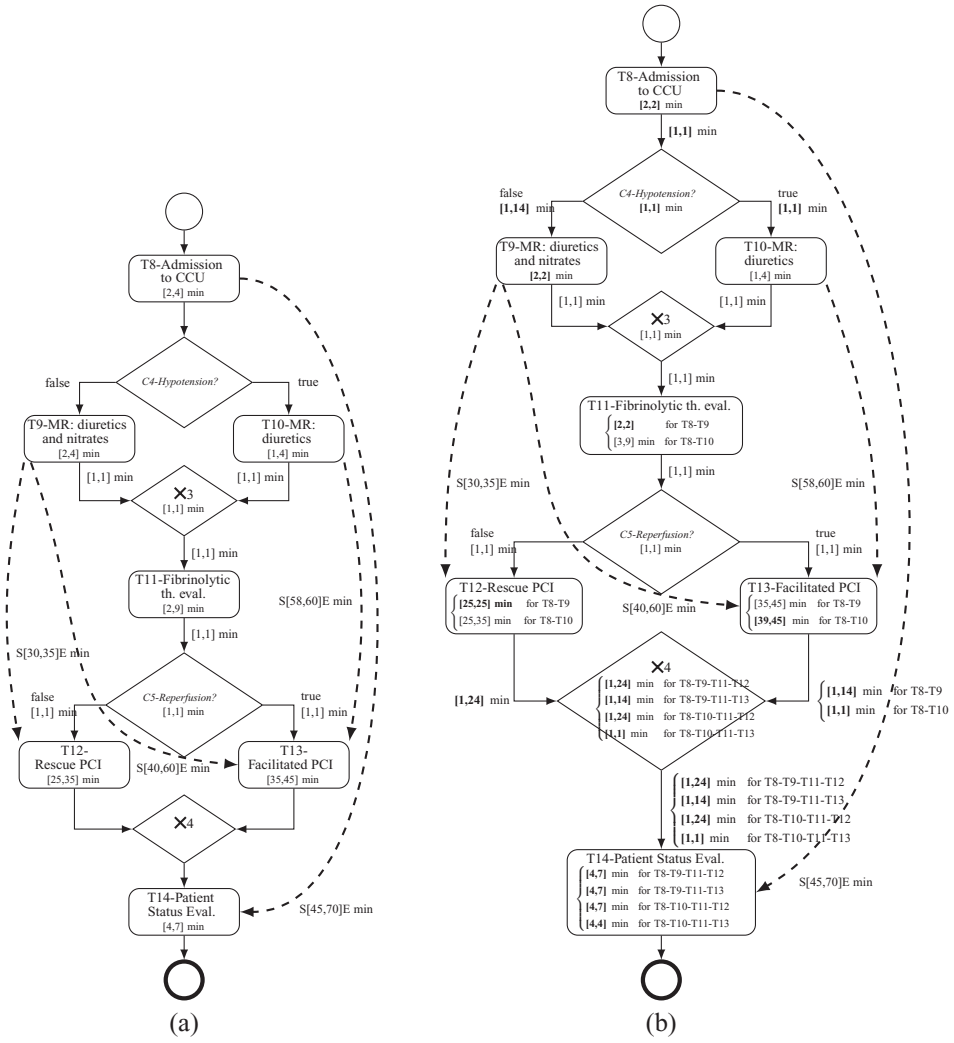



Fig. 8. History-dependent consistency case: (a) Original workflow graph schema with two relative constraints reduced (the boldface ones); (b) the result of consistency analysis. The schema is history-dependent consistent. The boldface ranges are different from the corresponding ones in (a).

set of task duration ranges to agents for each task and to itself for each connector. For each activity, exactly one of the given ranges could be used (enlarged or restricted) in accordance with the execution time of the previous completed activities. Therefore, the time required by an activity does not affect the following possible flows, but only the duration range for the following activities. It is worth noting that to check the history-dependent consistency it is sufficient to consider the last ✕ join connectors (there could be several last ✕ join connectors if they are on parallel flows) as we will discuss in the following.

An example of a history-dependent consistent workflow schema is given in Figure 8(a). This schema differs from the original schema of Figure 7(a) for the two relative constraint values: T9-T12 new value is S[30,35]E min and T10-T13 one is S[58,60]E min. This schema is not strongly consistent because the consistency check for the prefix of Start determines that task T11 has an empty duration range. So, prefixes of ✕4 are

considered (i.e., prefixes T8-T9-T11-T12, T8-T9-T11-T13, T8-T10-T11-T12, and T8-T10-T11-T13). Since they are consistent and all the considered activities/edges preceding ✕4 have nonempty common ranges for their respective prefixes, the schema is history-dependent consistent. Task T11 has two ranges of possible durations, that is, [2, 2] min and [3, 9] min: the former range is the result of deriving the common range for *wf-paths* of prefix T8-T9-T11, while the other one is the result considering the *wf-paths* of prefix T8-T10-T11. In this case, even though it is not possible to have any common range of durations for T11 at design time, we know that there are allowed durations for T11 not preventing any following flow. The same consideration could be done for task T8, having in this case a single prefix composed by four different *wf-paths*.

The less desirable consistency is the *weak* one.

*Definition* 3.6 (*Workflow Weak Consistency*). A workflow schema is *weakly consistent* if: (i) all of its *wf-paths* are consistent in isolation and (ii) it is not history-dependent consistent.

If a workflow is weakly consistent, a first consequence is that an instance could not end successfully. Indeed, there is at least one activity in a prefix having nonintersecting ranges for two different *wf-paths* of the prefix. In this case, it could be that the considered task takes a duration that prevents the execution of the path that the workflow has to follow.

Figure 9(a) represents the original workflow schema with two relative constraint values modified: the constraint between T9 and T12 is changed from S[30,40]E min to S[30,35]E min and the constraint T9-T13 is changed from S[40,60]E min to S[58,60]E min. Having modified these relative constraints, the calculated ranges for T11 change (Figure 9(b)): in particular, for the prefix T8-T9-T11 we have an empty intersection, as the two ranges for the *wf-paths* belonging to the prefix are [2,2] min and [3,9] min, respectively. It means that it is not possible in the prefix T8-T9-T11 to derive a duration range for task T11 allowing any possible following path, that is, either T12 or T13.

Since each *wf-path* is consistent in isolation, the workflow is weakly consistent.

The general algorithm `consistencyCheck(`$G$`)` determining the consistency type of a workflow is depicted in Figure 10. Given a workflow graph $G$, `consistencyCheck(`$G$`)` checks if each *wf-path* of $G$ is consistent and, if so, it tries to determine the temporal ranges that can be used in any *wf-path* for activities/edges following the starting point. If such common ranges are possible (i.e., they are not-empty and all *wf-paths* with such ranges are consistent), then the workflow is strongly consistent and the algorithm returns the strong consistency status with the new set of temporal ranges. Otherwise, the algorithm determines all last ✕-Join connectors in *wf-paths* and tries to find, for any of them, if there exists a temporal assignment satisfying all temporal constraints such that for each activity/edge $a$ that precedes the considered ✕-Join connector there exists a common temporal range for each of prefixes of $a$ (algorithm `consistencyCheck(`$G$`,` rootActivity`)` in Figure 11). ✕-Join connectors are considered because they distinguish different past histories: indeed, after an ✕-Join connector, activities/edges have more prefixes than activities/edges preceding it have, and with more prefixes it is possible to analyze the consistency of a workflow according to different histories, separately.

If there exists an ✕-Join connector such that all its prefixes are strongly consistent and every activity/edge preceding it has a common temporal range among its prefixes, then the workflow is history-dependent consistent.

If it is not possible to determine a history-dependent consistency, then the algorithm returns the weak consistent status because each *wf-path* is consistent.

The algorithm `consistencyCheck(`$G$`,` rootActivity`)` in Figure 11 determines whether all prefixes of the element rootActivity of the workflow $G$ are all strongly consistent

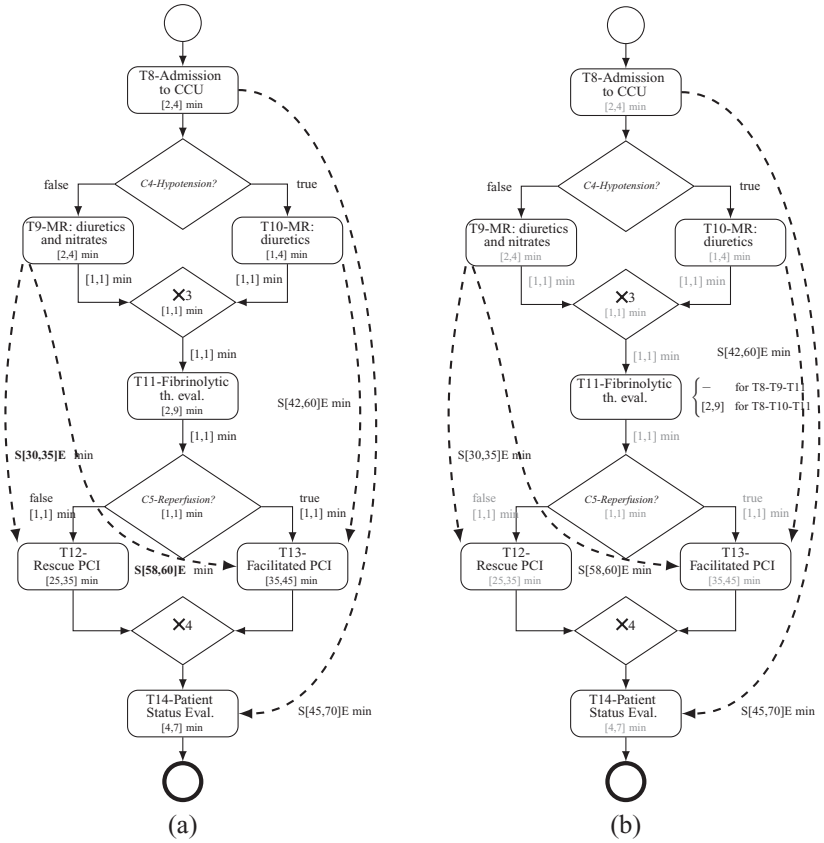



Fig. 9. Weak consistency case: (a) Original workflow graph schema with two relative constraints reduced (the boldface ones); (b) the result of consistency analysis. The schema is weakly consistent because the only task that has all its prefixes consistent is T14 and each of these prefixes has cardinality one. It is not possible to derive duration ranges for tasks ensuring a successful execution of any *wf-path* (so we use gray color reporting the original ranges).

and whether every activity/edge preceding rootActivity has a single temporal range for each of its prefixes. We recall that if the two conditions hold for a rootActivity $r$, it is possible to assign to each activity/edge preceding $r$ a temporal range that could depend from the history (i.e., the previously executed tasks), but that does not prevent any possible *wf-path* after the activity/edge.

A final note regards the time complexity of the workflow consistency check. In the previous section we state that the consistency check of a *wf-path* can be done in time $O(n^3)$, where $n$ is the number of nodes of the *wf-path* graph. Since `buildCommonRanges(`$S$`)` (Figure 6) makes a limited number of operations on the results of checking the consistency of *wf-paths*, the time complexity of the whole process can be determined by multiplying $O(n^3)$ by the number of possible *wf-paths* of $S$ and by the maximum number of iterations of the do-while cycle. Such maximum number, even though in our experimental tests results to be around the number 4, in the worst case analysis is limited by the maximum extension of temporal ranges present into *wf-paths* ($MaxRange$). Therefore, the time complexity of `buildCommonRanges(`$S$`)` is pseudopolynomial with respect to the cardinality of $S$, $n$ and $MaxRange$, that is, $O(n^3 \ |S| \ MaxRange)$. As regards `consistencyCheck(`$G$, rootActivity`)`, the complexity can be estimated by multiplying the complexity of `buildCommonRanges` by the number of possible prefixes of $rootActivity$

**Function** consistencyCheck($G$)

**Input**: $G$: workflow graph to analyze
**Output**: the consistency status of $G$ and a possibly new set of ranges for activities and delays
// Initial *wf-paths* consistency check
**foreach** *(wf-path* $p \in G$*)* **do**
  Execute Floyd-Warshall on the distance graph corresponding to $p$;
**if** *(at least one wf-path is inconsistent)* **then**
  **return** *('Not Consistent Workflow',* $\emptyset$*);*
// All *wf-paths* are consistent, determine the kind of consistency
$(status, R)$ =consistencyCheck($G$, Start);
**if** *(status ==='Strong Consistency Workflow')* **then**
  **return** *('Strong Consistency Workflow', R);*
// Determine the ✕ join connectors to analyze
✕JoinSet = $\{x \mid x$ is the last ✕ Join connector in a *wf-path* $\}$;
**foreach** $(x \in$ ✕*JoinSet*$)$ **do**
  $(status, R)$ =consistencyCheck($G, x$);
  **if** *(status ==='Strong Consistency Workflow')* **then**
    **return** *('History-Dependent Consistency Workflow', R);*
**return** *('Weak Consistency Workflow',* $\emptyset$*);*

Fig. 10. ConsistencyCheck($G$) determines whether a workflow schema is consistent and, if so, the kind of consistency by means of algorithm consistencyCheck($G$, $rootActivity$). Moreover, it returns the set of temporal ranges can be used as initial ranges to start the workflow execution.

and by the number of iterations of the do-while cycle. The number $K_{rootActivity}$ of possible prefixes of rootActivity is given by counting all possible flows from Start to rootActivity. 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 rootActivity ($MaxRange$). Thus the overall complexity of `consistencyCheck(`$G$, rootActivity`)` can be expressed as $O(K_{rootActivity} \; |S_{rootActivity}| \; n^3 \; MaxRange^2)$ where $S_{rootActivity}$ is the average cardinality of prefixes of rootActivity, observing that the product $K_{rootActivity} \; |S_{rootActivity}|$ corresponds to the total number of *wf-paths* of all prefixes of rootActivity. Denoting by $w$ the number of *wf-paths* of the graph, it holds that $K_{rootActivity} \; |S_{rootActivity}| \leq w$ and so the `consistencyCheck(`$G$, rootActivity`)` complexity can be approximated as $O(w \; n^3 \; MaxRange^2)$.

Finally, the time complexity of `consistencyCheck(`$G$`)` is given by the order of complexity of `consistencyCheck(`$G$, rootActivity`)` multiplied by the dimension of ✕JoinSet. An estimation of the (worst-case) dimension of ✕JoinSet is given assuming that the last ✕Join connectors are on a parallel path (each on a different branch) and their degree is 2: in this case the dimension of ✕JoinSet is less than the half of the number of *wf-paths* of the graph. Therefore the time complexity is $O(w^2 \; n^3 \; MaxRange^2)$.

This last number $w$ depends on the number of alternative or conditional or cycle connectors in the workflow schema and by their relative positions. An upper bound can be easily determined assuming that all of these connectors are not nested, but are in sequence. For each alternative connector, there are as many *wf-paths* as the outgoing edges of the connector. For each conditional connector, there are two *wf-paths*. For each cycle, there are as many *wf-paths* as the maximum number of iterations of the cycle. Labeling by $c$ the total number of conditional connectors, by $a = \max\{|a_l \star|\}$ the maximum among the outdegrees of the alternative connectors and by $m$ the maximum number of iterations for the cycles of the schema, an estimation of the number of

**Function** consistencyCheck($G$, *rootActivity*)

**Input**: $G$: a consistent workflow graph; *rootActivity*: activity from which to start the analysis
**Output**: the consistency type of $G$ and a new set of ranges for activities and delays
$P$ = set of all prefixes of *rootActivity*;
// $R_p$ = array of temporal 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 $G$;
**do**
  **foreach** $\{prefix \mid R_{prefix} \neq null\}$ **do** // Save last found solution
    $R'_{prefix} = R_{prefix}$;
  **foreach** *prefix* $\in P$ **do** // Check if the prefix is strongly consistent
    $(status, R_{prefix})$ =buildCommonRanges(*prefix*);
    **if** *(status* $\neq$ *'commonRangesFound')* **then**
      // It is necessary to evaluate another *rootActivity*
      **return** *('Non-Strong Consistency Workflow',* $\emptyset$*);*
  **foreach** $\{a \mid a$ *is an activity/edge* $\wedge$ *a precedes rootActivity*$\}$ **do** // Determine common range among prefixes for each element preceding *rootActivity*
    $P' = \{p' \mid p'$ is prefix of $a\}$;
    **foreach** *prefix*$' \in P'$ **do** // Determine the common range among prefixes for $a$
      $R_{prefix'}[a]$ = intersection of all $a$-ranges present in *wf-paths* of *prefix'*;
      **if** $(R_{prefix'}[a] == \emptyset)$ **then**
        **return** *('Non-Strong Consistency Workflow',* $\emptyset$*);*
      **foreach** $\{prefix'' \mid prefix' \preceq prefix''\}$ **do** // Propagate the new range to all prefixes greater than *prefix'*
        $R_{prefix''}[a] = R_{prefix'}[a]$;
**while** $(\exists$*prefix of activity/edge b s.t.* $R_{prefix}[b] \neq R'_{prefix}[b])$;
**return** *('Strong Consistency Workflow',* $\cup_{prefix} R_{prefix}$*);*

Fig. 11. ConsistencyCheck($G$, *rootActivity*) determines whether all prefixes of the element rootActivity of the workflow $G$ are all strongly consistent and whether each activity/edge preceding rootActivity has a single temporal range for each of its prefixes. If so, it returns the set of temporal ranges can be used as initial ranges for the prefixes.

*wf-paths* is $1 \leq w \leq 2^c(a+1)^j(m+1)^i$, where $j$ is the number of alternative connectors and $i$ is the number of cycle constructs. It's worth to note that the number of *wf-paths* can be exponential with respect to the graph order even if the schema admits only one cycle construct ($i = 1$) and there are no alternative split or conditional operators ($j = 0 \wedge c = 0$) because $m$ can assume a exponential value with respect to the dimension of the instance.

In summary, the upper bound to the time complexity of the workflow consistency check is exponential with respect to the number of conditional/alternative/cycle connectors and to the dimension of width of temporal ranges.

### 3.5. Runtime Temporal Workflow Consistency

During the execution, a temporal *WfMS* has also to verify that the considered (partial) case can reach its end in a consistent way, that is, satisfying all the given temporal constraints. To do that, the *WfMS* has to check the consistency of the workflow graph obtained from the original one by updating the durations of completed activities and

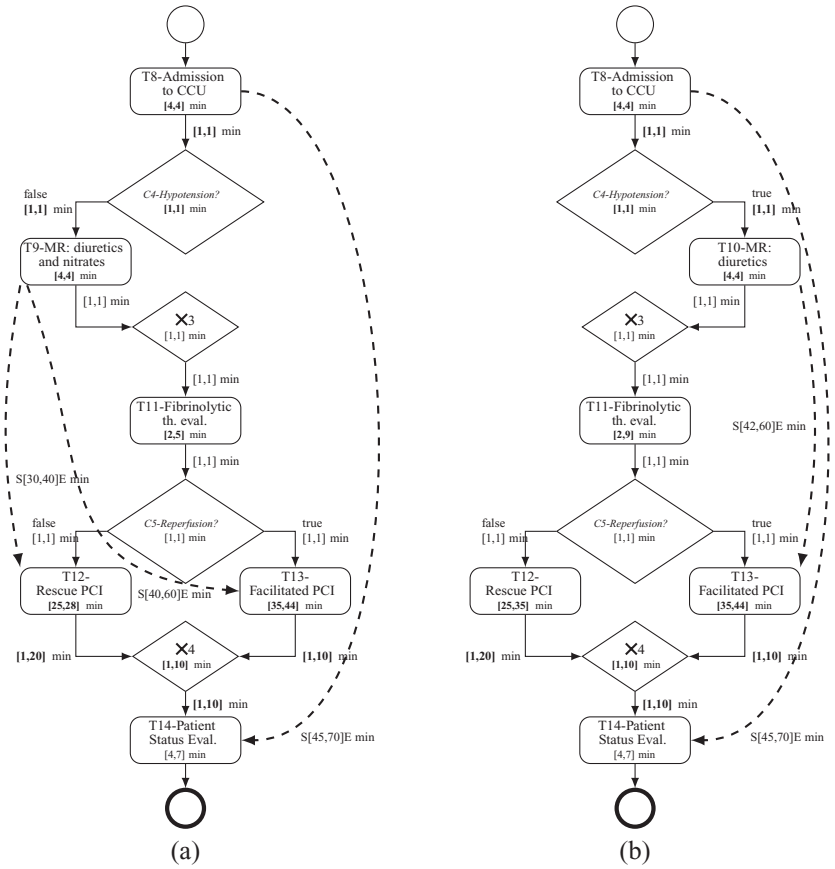



Fig. 12. Strong consistency runtime case: (a) Workflow schema after the execution of tasks *wf-path* T8-T9 assuming that T8 and T9 last their maximum allowed time while all delays and connectors among these activities last the minimum time; (b) workflow schema after the execution of tasks *wf-path* T8-T10 with analogous assumptions about tasks, connectors, and delays. Boldface label represents the value changed with respect to the original one.

delays to their actual values. Moreover, the workflow graph has to be pruned of the *wf-paths* that have not been followed.

In the following, we discuss the computational issues for the runtime evaluation of workflow consistency according to the consistency classification that we have proposed in Section 3.4.

*3.5.1. Runtime Checking Strongly Consistent Workflows.* A first benefit of strong consistency is that at the start of an execution the preliminary allowed range for each task is available to the *WfMS* either for internal evaluation of workload balancing among agents or for transmission to the (human) agents involved with workflow execution. During the execution, the *WfMS* has to manage the change of the recomputed allowed ranges: these ranges could be restricted or enlarged depending on the actual duration of the already executed activities and on the excluded *wf-paths* (e.g., due to previous alternative/conditional splits).

Let us consider how the WfMS has to deal with runtime consistency by considering the execution of the workflow schema obtained after the evaluation of design-time strong consistency, depicted in Figure 7(b).

In Figure 12 we show two different executions: in subfigure (a) we consider the runtime consistency evaluation just before the execution of task T11 after the execution of activities T8, C4, T9, and X3, while in subfigure (b) we consider the runtime consistency evaluation just before the execution of task T11 and after the execution of activities T8, C4, T10, and X3. In these executions tasks required their maximum allowed time, while the connectors and delays used the minimum one. It is worth to observe that the new duration ranges for tasks T11 and T12 are restricted with respect to the ranges computed at design time when T11 follows T8 and T9: indeed, ranges [2,7] min and [25,30] min for tasks T11 and T12 are restricted to [2,5] min and [25,28] min, respectively. On the other side, when T11 follows the execution of T8 and T10, the new duration ranges for tasks T11 and T12 are enlarged and assume the new values [2,9] min and [25,35] min, respectively.

*3.5.2. Runtime Checking History-Dependent Consistent Workflows.* With a history-dependent consistent workflow schema, more flexibility is required either to the *WfMS* or to the agents because for each task there can be more allowed preliminary disjoint ranges. In particular, for the *WfMS* it is more complex to manage the workload balancing due to the presence of (possibly) several ranges for each activity. As in the strong consistency case, during the execution the *WfMS* has to manage the restriction or enlargement of the allowed ranges; moreover, it has to manage the changes about the number of possible ranges for each activity. It is worth to note that at some point during the workflow execution, the number of temporal ranges for each activity decreases to one and, therefore, the remaining partial schema becomes strongly consistent.

Let us consider how the *WfMS* has to deal with runtime consistency by considering the execution of the workflow schema obtained after the evaluation of design-time history-dependent consistency, depicted in Figure 8(b). In this case tasks T11, T12, T13, and T14 have different allowed ranges according to their prefixes.

In Figure 13 we show two different executions: in subfigure (a) we consider the runtime consistency evaluation just before the execution of task T11 after the execution of activities T8, C4, T9, and X3, while in subfigure (b) we consider the runtime consistency evaluation just before the execution of task T11 after the execution of activities T8, C4, T10, and X3. In these executions, tasks required their maximum allowed time, while connectors and delays used the minimum one. It is worth to observe that for task T11, the allowed range remains the same computed at design time. Moreover, when evaluating the consistency of prefixes T8-T9-T11 and T8-T10-T11 separately, both the two corresponding workflow schemata result to be strongly consistent: it means that when the *WfMS* is ready to assign the execution of task T11, it is also able to derive the (preliminary) single temporal range for each activity following T11. Any of these single temporal ranges depends only on the execution history preceding task T11: for example, for task T13, it is possible to derive the range [35, 45] min when T8-T9 are executed before T11, while the interval [39, 45] min is derived when T8-T10 are executed before T11.

*3.5.3. Runtime Issues with Weakly Consistent Workflows.* Weak consistency does not guarantee the successful execution of any *wf-path*. In particular, the *WfMS* has to manage suitable exceptions when any task needs to be executed and there is no allowed nonempty range for it.

For example, considering the weak consistency case depicted in Figure 9, any execution involving tasks T8 and T9 determines that T11 has no allowed temporal range to be executed without preventing any possible further *wf-path*.

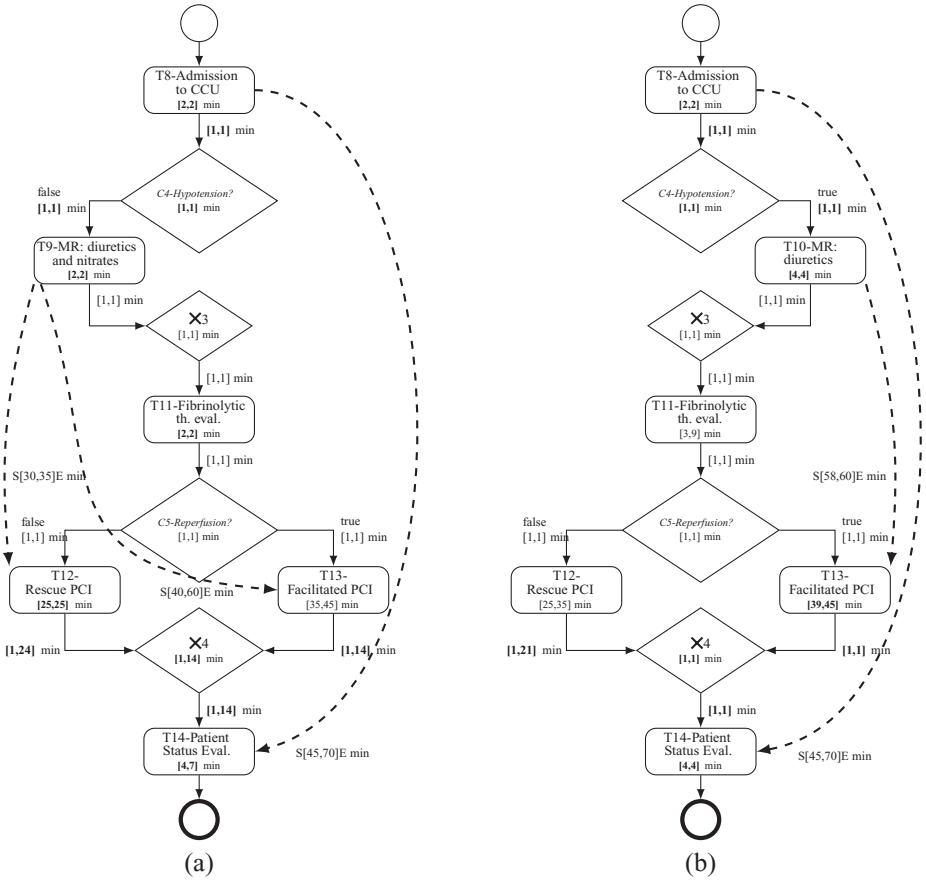



Fig. 13. History-dependent consistency runtime case: (a) Workflow schema after the execution of tasks *wf-path* T8-T9 assuming that T8 and T9 last their maximum allowed time while all delays and connectors among these activities last the minimum time; (b) workflow schema after the execution of tasks *wf-path* T8-T10 with analogous assumptions about tasks, connectors, and delays. Boldface label represents the value changed with respect to the original one.

## 4. RELATED WORK

Many research papers have been published and several approaches for time modeling have been proposed. In this section we compare our conceptual model with other similar conceptual models from the literature. In order to summarize the main aspects of the other approaches, Table I shows a compact comparison between the main features of our conceptual model and the corresponding features of the other proposals on temporal workflows.

Eder and colleagues started to investigate on temporal constraints in workflows since the end of nineties and in the last years they focused on applying and extending the proposed techniques in specific workflow-related domains, such as that of coreographies and interorganizational workflows [Eder et al. 1999, 2000, 2007, 2008; Bierbaumer et al. 2005; Eder and Tahamtan 2008a, 2008b; Pichler et al. 2009]. Basically, these proposals are based on a structure named Timed Workflow Graph (*TWG*), allowing one to represent temporal properties of tasks, named in this context "activity nodes". TWG is a Directed Acyclic Graph (DAG), in which nodes are activities and oriented edges are control flows. DAGs are not not suitable to represent complex workflow processes and

Table I. Compact Comparison between the Main Features of Our Conceptual Model and the Corresponding Features of the Other Proposals in the Literature

| | Our model | TWG | Marjanovic's model | Bettini's model | Chen's model |
|---|---|---|---|---|---|
| WfMC compliance | yes | yes | yes | yes, partially | yes |
| Cycles representation | yes | no | no | no | no |
| Tasks-connectors distinction | yes | no | yes, partially | no | no |
| Ranges for durations | yes | no | yes | yes | yes |
| Delay specification | yes | yes | no | yes | yes |
| Relative constraints | yes | yes | yes | yes | yes |
| Unbounded intervals | yes | no | no | yes | no |
| Consistency check algorithm | yes | yes | yes | yes | yes |
| Consistency classification | yes | no | no | yes, partially | yes |

iterative executions, such as cycles, cannot be mapped by DAGs. The TWG approach is based on the Critical Path Method (CPM) [Taha 2010] extended to consider explicit time constraints between start and end of different activities [Eder et al. 1999] and, then, to consider in a proper way the presence of conditional branches in a workflow schema. CPM [Taha 2010] is a well-known method used in project planning to identify the overall time length of a project and the activities that are in the critical path, that is, the path that mainly determines the overall length of the project. Eder and colleagues in Eder et al. [1999; 2000] focused on introducing temporal constraints and alternative paths, but did not consider the issue related to the fact that activities may have a set of allowed durations, that cannot be fixed at design time and that could require a runtime evaluation to "tune" the initial temporal workflow schema to the actual workflow execution: indeed, being based on CPM, each task of TWG is described by one single value of duration. Recently, in Eder and Tahamtan [2008a, 2008b], Eder et al. [2008], and Pichler et al. [2009] different applications and extensions have been provided to TWG: in particular, in Eder and Tahamtan [2008a, 2008b] temporal consistency methods are proposed for federated choreographies and interorganizational workflows, while in Eder et al. [2008] an extension of TWG is proposed, dealing with possibly different task durations, where both task durations and conditional paths are characterized by probability functions. Finally, TWG does not provide multiple granularity management: all the temporal attributes are expressed by absolute integers representing a number of workflow temporal units.

Marjanovic et al. define in Marjanovic and Orlowska [1999] a conceptual model that classifies temporal aspects of workflow schemata as: "basic temporal constraint", "limited duration constraint", "deadline constraint", and "interdependent temporal constraint". A basic temporal constraint limits the expected duration of one single task: in our model the duration is specified by a range, represented by the minimum and maximum durations. A limited duration constraint is an upper bound for the duration of the workflow execution: in our proposal it can be expressed by a relative constraint between the Start node and each End node of the workflow graph. A deadline constraint, in terms of absolute time, limits when a task can start or complete during a workflow execution: this constraint corresponds to the absolute constraint of our model. An interdependent temporal constraint limits the time distance between two tasks in a workflow model: it has the same meaning of the relative constraint in our model. Marjanovic and Orlowska's [1999] model does not permit any delay interval between consecutive activities: the completion instant of a task corresponds to the starting instant of the subsequent task.

The model by Bettini et al. [2002b] is quite different from the previous ones. First of all, the nodes of a workflow graph are not tasks but correspond to temporal instants. Every task is represented by two nodes: the starting instant and the ending instant. Every edge in the workflow graph represents the temporal distance between two nodes:

edge label is an interval representing the time distance between the connected nodes. If a pair of nodes represents the starting and the ending instants of a task, the label on the connecting edge is the allowed duration of the task. In the same way, if a pair of nodes represents the ending instant of a task and the starting instant of another task, the connecting label edge represents the allowed delay between the first task execution ending instant and the second task execution starting instant. Bettini et al. [2002b] do not consider the issue of consistency for multiple *wf-paths*, while they focus of the consistency check for temporal workflow schemata with multiple granularities, showing that it is a NP-complete problem [Bettini et al. 2002a]. Indeed, in their approach the conversion of a graph of interval constraints for activities at different granularities into a graph with all the constraints at the same (finest) granularity produces a graph having more than one temporal interval constraint between nodes; this kind of graph can be checked only by using exponential-time algorithms unless P=NP [Dechter et al. 1991]. It is worth noting that granularity conversions in our approach are simply conversions among time units and we do not consider more sophisticated granularity conversions [Bettini et al. 2002b; Goralwalla et al. 2001].

In Chen and Yang [2007], Chen and Yang focus on specific algorithms for selecting at runtime some suitable checkpoints allowing the verification of temporal constraints on grid workflows. As the focus of this article is on runtime evaluation of temporal constraints, the proposal from Chen and Yang [2007] could be considered as complementary to our conceptual model. Nevertheless, it is interesting to highlight some differences in the concept of temporal consistency that the authors introduce to motivate their proposal. They propose the concepts of strong consistency and weak consistency of a constraint and consider only upper bound constraints: a temporal constraint between two activities (i.e., between their ends, according to the approach in Eder et al. [1999]) is strongly consistent if the maximum time distance between the ends of the given activities, evaluated by considering the past execution history, is less than the given value for the temporal constraint. A temporal constraint is weakly consistent if its value is between the mean time distance and the maximum time distance for the given tasks, on the base of the past execution history.

As this approach is based on TWGs, proposed in Eder et al. [1999, 2000], there is not a clear distinction between task constraints and edge/connector constraints: all the delays managed by the *WfMS* are considered as part of the duration of the next tasks. Thus, there is not a clear distinction between constraints that the agents have to manage in executing the assigned activities and constraints which affect the (internal) activities of the *WfMS*. In Chen and Yang [2007], constraints are given at a single basic time unit.

## 5. XML REPRESENTATION OF WORKFLOW GRAPHS AND A PROTOTYPE IMPLEMENTATION

The WfMC Reference Model [WfMC 1995] can be described in terms of the XML-Schema proposed by the WfMC [WfMC 2002]. We have extended this schema by adding new modeling constructs so that a *WfMS* capable of managing the XML descriptions from the WfMC may also deploy the XML schema based on our conceptual model exploiting our time constructs. More details are given in online Appendix B.

Finally, we have implemented an extension to the well-known WfMS YAWL [van der Aalst et al. 2004] to support workflow modeling and time management at workflow design time. The reason for developing such an extension is twofold. First, it is the creation of a tool supporting the design and the verification of temporal workflow schemata. Second, it is to show, as a proof-of-concept, that our theoretical and methodological proposal can be realized in a tool supporting the specification of real-world workflow schemata. More details about our implementation are in online Appendix C.

## 6. CONCLUSIONS

Time is a fundamental concept of every workflow process, and procedures for checking temporal workflow consistency are a crucial component of a WfMS. At the best of our knowledge, currently there are no proposals for temporal modeling of complex workflow processes, whose schema temporal consistency is checked.

In this article, we introduced a powerful temporal conceptual model, which extends in a seamless way the basic model proposed by the WfMC: it provides constructs to formalize complex processes, such as conditional or parallel executions and cycles. The specification of a large number of temporal constraints enables the designer to describe temporal properties of tasks. The constraints can refer to either relative or absolute times. The temporal constraints belonging to the same workflow schema can be expressed using different time granularities. We also proposed different kinds of workflow temporal consistency and discussed how to check them at design time; moreover, we described how these consistencies require different levels of flexibility from the *WfMS* at runtime. All the components of this model have also been represented by an XML-Schema document, which extends the XML-Schema specification proposed by the WfMC. A proof-of-concept prototype supporting our conceptual model has been realized extending the YAWL WfMS.

## ELECTRONIC APPENDIX

The electronic appendix to this article can be accessed in the ACM Digital Library.

## REFERENCES


ANTMAN, E. M., ANBE, D. T., ARMSTRONG, P. W., BATES, E. R., GREEN, L. A., HAND, M., HOCHMAN, J. S., ET AL. 2004. ACC/AHA guidelines for the management of patients with ST-elevation myocardial infarction. *Circul. 110*, 5, 588–636.

BETTINI, C., WANG, X. S., AND JAJODIA, S. 2002a. Solving multi-granularity temporal constraint networks. *Artif. Intell. 140*, 1-2, 107–152.

BETTINI, C., WANG, X. S., AND JAJODIA, S. 2002b. Temporal reasoning in workflow systems. *Distrib. Parallel Datab. 11*, 3, 269–306.

BIERBAUMER, M., EDER, J., AND PICHLER, H. 2005. Accelerating workflows with fixed date constraints. In *Proceedings of the International Conference on Conceptual Modeling (ER'05)*, L. M. L. Delcambre, C. Kop, H. C. Mayr, J. Mylopoulos, and O. Pastor, Eds., Lecture Notes in Computer Science, vol. 3716, Springer, 337–352.

CHEN, J. AND YANG, Y. 2007. Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in grid workflow systems. *ACM Trans. Auton. Adapt. Syst. 2.*

COMBI, C. AND POZZI, G. 2003. Temporal conceptual modelling of workflows. In *Proceedings of the International Conference on Conceptual Modeling (ER'03).* I.-Y. Song, S. W. Liddle, T. W. Ling, and P. Scheuermann, Eds., Lecture Notes in Computer Science, vol. 2813, Springer, 59–76.

COMBI, C. AND POZZI, G. 2004. Architectures for a temporal workflow management system. In *Proceedings of the ACM Symposium on Applied Computing (SAC'04).* H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, Eds., ACM Press, New York, 659–666.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms.* The MIT Press.

DECHTER, R., MEIRI, I., AND PEARL, J. 1991. Temporal constraint networks. *Artif. Intell. 49*, 1-3, 61–95.

EDER, J. AND GRUBER, W. 2002. A meta model for structured workflows supporting workflow transformations. In *Proceedings of the International Conference on Advances in Databases and Information Systems (ADBIS'02).* Y. Manolopoulos and P. Navrat, Eds., Lecture Notes in Computer Science, vol. 2435, Springer, 326–339.

EDER, J., GRUBER, W., AND PANAGOS, E. 2000. Temporal modeling of workflows with conditional execution paths. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA'00).* M. T. Ibrahim, J. Kung, and N. Revell, Eds., Lecture Notes in Computer Science, vol. 1873, Springer, 243–253.

EDER, J., LEHMANN, M., AND TAHAMTAN, A. 2007. Conformance test of federated choreographies. In *Proceedings of the International Conference on Interoperability for Enterprise Software and Applications (IESA'07).* R. Jardim-Goncalves, J. P. Muller, K. Mertins, and M. Zelm, Eds., Springer, 223–234.

EDER, J. AND PANAGOS, E. 2000. Managing time in workflow systems. In *Workflow Handbook 2001*, Workflow Management Coalition (WfMC), 109–132.

EDER, J., PANAGOS, E., AND RABINOVICH, M. 1999. Time constraints in workflow systems. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'99)*. M. Jarke and A. Oberweis, Eds., Lecture Notes in Computer Science, vol. 1626, Springer, 286–300.

EDER, J., PICHLER, H., AND TAHAMTAN, A. 2008. Probabilistic time management of choreographies. In *Proceedings of the Business Process Management Workshops*. D. ardagna, M. Mecella, and J. Yang, Eds., Lecture Notes in Business Information Processing, vol. 17, Springer, 443–454.

EDER, J. AND TAHAMTAN, A. 2008a. Temporal conformance of federated choreographies. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA'08)*. S. S. Bhowmick, J. Kung, and R. Wagner, Eds., Lecture Notes in Computer Science, vol. 5181, Springer, 668–675.

EDER, J. AND TAHAMTAN, A. 2008b. Temporal consistency of view based interorganizational workflows. In *Proceedings of the International United Information Systems Conference (UNISCON'08)*. R. Kaschek, C. Kop, C. Steinberger, and G. Fliedl, Eds., Lecture Notes in Business Information Processing, vol. 5, Springer, 96–107.

GORALWALLA, I. A., LEONTIEV, Y., OZSU, M. T., SZAFRON, D., AND COMBI, C. 2001. Temporal granularity: Completing the puzzle. *J. Intell. Inf. Syst. 16*, 1, 41–63.

ISO. 2004. ISO 8601:2004: Representation of dates and times. ISO. http://www.iso.org

MARJANOVIC, O. AND ORLOWSKA, M. E. 1999. On modeling and verification of temporal constraints in production workflows. *Knowl. Inf. Syst. 1*, 2, 157–192.

NURCAN, S. 2008. A survey on the flexibility requirements related to business processes and modeling artifacts. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS'08)*. IEEE Computer Society, 378.

OBJECT MANAGEMENT GROUP. 2007. Business process definition metamodel (bpdm), beta 1. http://www.omg.org

PICHLER, H., WENGER, M., AND EDER, J. 2009. Composing time-aware web service orchestrations. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'09)*. P. van Eck, J. Gordijn, and R. Wieringa, Eds., Lecture Notes in Computer Science, vol. 5565, Springer, 349–363.

RINDERLE, S., REICHERT, M., AND DADAM, P. 2004. Flexible support of team processes by adaptive workflow systems. *Distrib. Parallel Datab. 16*, 1, 91–116.

TAHA, H. A. 2010. *Operations Research: An Introduction* $9^{th}$ Ed. Prentice-Hall.

VAN DER AALST, W. M. P., ALDRED, L., DUMAS, M., AND TER HOFSTEDE, A. H. M. 2004. Design and implementation of the YAWL system. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'04)*. A. Persson and J. Stirna, Eds., Lecture Notes in Computer Science, vol. 3084, Springer, 142–159.

VAN DER AALST, W. M. P., HIRNSCHALL, A., AND VERBEEK, H. M. W. E. 2002. An alternative way to analyze workflow graphs. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'02)*. A. B. Pidduck, J. Mylopoulos, C. C. Woo, and M. T. Ozsu, Eds., Lecture Notes in Computer Science, vol. 2348, Springer, 535–552.

VAN HEE, K. M., OANEA, O., SEREBRENIK, A., SIDOROVA, N., VOORHOEVE, M., AND LOMAZOVA, I. A. 2007. Checking properties of adaptive workflow nets. *Fundam. Inf. 79*, 3-4, 347–362.

WFMC. 1995. The workflow reference model. http://www.wfmc.org/reference-model.html

WFMC. 2002. Workflow process definition interface- XML process definition language. Tech. rep., Workflow Management Coalition. http://www.wfmc.org

Received February 2010; revised September 2010; accepted February 2011