Knowledge and Information Systems 1 (1999) 157–192

Knowledge and Information Systems

On Modeling and Verification of Temporal Constraints in Production Workflows

Olivera Marjanovic and Maria E. Orlowska

Department of Computer Science and Electrical Engineering
University of Queensland, QLD 4072, Brisbane, Australia
E-mail: o.marjanovic@mailbox.uq.edu.au, maria@csee.uq.edu.au

Received 10 March 1998
Revised 10 August 1998
Accepted 26 September 1998

Abstract. The dynamic nature of events, in particular business processes, is a natural and accepted feature of today's business environment. Therefore, workflow systems, if they are to successfully model portions of the real world, need to acknowledge the temporal aspect of business processes. This is particularly true for processes where any deviation from the prescribed model is either very expensive, dangerous or even illegal. Such processes include legal processes, airline maintenance or hazardous material handling. However, time modeling in workflows is still an open research problem. This paper proposes a framework for time modeling in production workflows. Relevant temporal constraints are presented, and rules for their verification are defined. Furthermore, to enable visualization of some temporal constraints, a concept of "duration space" is introduced. The duration algorithm which calculates the shortest/longest workflow instance is presented. It is a generalization of two categories of algorithms: the shortest-path partitioning algorithm and the Critical Path Method (CPM). Based on the duration algorithm, the verification algorithm is designed to check the consistency of introduced temporal constraints.

Keywords: Workflows systems, time modeling, temporal constraints, verification.

1 Introduction

Workflow systems have been identified as the latest in a long line of key enablers of dynamic and responsive organisations. The evidence for their growing

acceptance may be found in numerous application domains, such as finance and banking, office automation, health care, software engineering and telecommunications [30]. Workflows are defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules" [31]. They are usually classified either as *ad hoc, administrative* or *production* workflows [11]. In this paper, we will limit our discussion to production workflows which involve repetitive and predictable business processes, such as, for example, loan applications or the processing of insurance claims. They typically encompass complex information processes involving access to multiple, heterogeneous data sources and information systems.

In spite of the extensive research and proliferation of commercial products, the requirements for workflows still far exceed the capabilities provided by products today [30]. There are still many open research problems that need to be addressed. One of them is time modeling and management [2, 3, 14, 17, 24]. Moreover, according to Geppert et al. [10], primitive support for time management has been identified as one of the most significant limitations of today's workflow management systems. This is a fundamental problem due to the fact that all business processes exist in a temporal context and are time constrained.

However, currently, there is no standard addressing time modeling for production workflows. The Workflow Management Coalition Report [31] whose aim is to establish a consistency in the specification and use of workflow terminology, does not contain any specification for time modeling. Very few workflow research projects take into consideration the general problem of time specification and management. For example, the ADEPT project [4] investigates modeling of the real-time deadline constraints and the consequences of missing deadlines in the case of structural changes of a workflow during its execution (e.g. introduction of new task in a workflow instance). However, this work is limited to deadline constraints only in the context of workflow execution and it does not examine the verification of temporal consistency. Another example is the HOST (Healthcare Open Systems and Trials) project [12]. In that project, the temporal reasoner called Tachyon is developed for time management in health-care processes. This work is limited to health-care processes only, and a temporal model upon which temporal reasoning is based is very simple, i.e. it does not include decision nodes nor modeling and verification of other temporal constraints apart from duration of individual tasks.

On the other hand, the state of the art in workflow systems is largely dictated by product vendors. The commercial products currently available offer very limited support for time representation and management. This appears mainly in the form of "calendars" and "to do" lists that specify deadlines for various tasks and generate alarms when deadlines are going to be missed. However, consistency of the deadline constraints and the consequences for missing certain deadlines are not known to the workflow management system [4].

The need for time management in production workflows can be illustrated by the following points:

- Early detection of time inconsistency of a workflow model will enable a user to predict any time-related problems, such as possible bottlenecks and any violations of temporal constraints. This is particularly important for processes where any deviation from the prescribed model can be either very expensive, dangerous or even illegal, for instance airline maintenance, hazardous material handling or legal and medical procedures.
- Often, due to unrealistic time estimates, costly *ad hoc* adjustments have to be made during workflow execution [6]. Proper time modeling simulation and the evaluation of workflow models will enable more realistic time estimates and more reliable workflow models.
- Time monitoring during workflow execution is very important for the identification and prediction of possible problems. For example, where a task is preceded or succeeded by exceptionally long transfer times, further investigation may reveal that bottlenecks have occurred.
- When unforeseen emergencies and organizational delay occur, time simulation may be used to find alternative paths of workflow execution, i.e. to reschedule future tasks, as well as to find possible ways to reduce or to compensate for future delays.
- Proper time modeling will open possibilities for the better coordination of individual tasks and better planning of business processes.
- Finally, the purpose of time modeling in workflows is not to control and restrict a workflow user in any way or to manage their time. Rather, the purpose is to provide workflow users with simple but powerful mechanisms that will help them to improve process efficiency and effectiveness.

This paper investigates the problem of time modeling in production workflows with the following objectives:

- to provide a foundation for time management in general, for production workflows,
- to propose a new approach to time modeling that will include identification, classification and formal modeling of various temporal constraints as well as verification of their temporal consistency,
- to provide a workflow user with an expressive and simple-to-use visualization concept that will enable specification and visual verification of temporal constraints.

The paper is structured as follows: Related work on time management in project management, job-shop scheduling and artificial intelligence is briefly described in Section 2. Section 3 introduces a workflow model used in this paper. Important time issues for time modeling in production workflows, such as relative and absolute time, temporal constraints and their temporal consistency, are discussed in Section 4. A basic temporal constraint called the duration constraint and a concept of the duration space are introduced in Section 5. A concept of a workflow instance type and its duration is presented in Section 6 along with the duration algorithm for calculation of the longest/shortest workflow instance type. Other relevant temporal constraints and rules for verification of their temporal

consistency are described in Section 7. The verification algorithm is presented in Section 8. Finally, our approach is illustrated by an example presented in Section 9.

2 Time Management—Related Work

Problems of time representation and management have been researched in other disciplines for many years. Project management is probably the most influential area in the field because of its well understood and accepted time representation and scheduling features such as the PERT chart, Gantt chart and CPM (Critical Path Method). There is a tendency within the workflow community to use project management concepts, and in particular, project management software for time management in workflows. The evidence can be found in both research and commercial sectors (see for example [13]). Although workflows and projects share many characteristics, e.g. they are time and resource constrained, planned, executed and controlled, we argue that production workflows significantly differ from projects and they require "workflow-specific" time management. The main differences can be summarized as follows. According to the Project Management Institute Standards Committee [23]: "A project is a temporary endeavor undertaken to create product or service". Temporary means that every project has a definite beginning and definite end, and it is not an ongoing effort. When uncertainty in a project drops to nearly zero, and when it is repeated a large number of times, the effort is usually no longer considered to be a project [21] but rather an operation. On the other hand, production workflows involve ongoing, highly predictable and repetitive business processes. Project models (or plans) are very general and they are frequently changed during project duration. Project plans do not include alternative paths, nor a concept of an instance. On the other hand, workflow models are much more detailed and therefore much more complex. Firstly, they include various workflow execution structures such as alternative, concurrent, exclusive "or-join" and synchronizer. Secondly, a workflow model can be seen as a collection of alternative instance types where a type is determined by a set of decision parameters. Thus, based on the same workflow model, several instances can be executed at the same time, being at different stages of their execution and progressing along different executive paths. The differences between project time management and workflow time management systems are discussed in more detail in [16].

Another related area, in many respects similar to project management, is job-shop scheduling. In general terms, job-shop scheduling requires completing a set of tasks while satisfying temporal and resource constraints. Temporal constraints determine the order of tasks while resource constraints determine which tasks can or cannot be done simultaneously (e.g. the same machine cannot perform two tasks at once). The main objective of job-shop scheduling is to create a schedule specifying when each task is to begin and finish and what resources it will use [27]. Although there are examples of the applications of job-shop scheduling to administrative office procedures (see for example [9]) job-shop scheduling and

workflow management are fundamentally different approaches to process modeling. While the former is the most suitable for so called material processes (i.e. engineering/manufacturing types of processes), the latter is designed to support business processes. For distinctions among material, information and business processes see [18]. By applying job-shop scheduling principles and algorithms to the modeling of business processes, one could oversimplify business processes to that of "head-down" document processing which has been a popular approach of the first office automation systems. It is important to point out that in workflow environments in the 90s the emphasis is on better educated workers who combine structured work with opportunity-based initiatives [18]. Thus,during workflow execution, users make various decisions creating different workflow instances, and these features make workflow management much more complex. Therefore, in workflows not all scheduled tasks will be performed as in job-shop scheduling. Rather, based on decisions made, some tasks will be selected while some may not be performed at all. Furthermore, allocation of tasks to individual users is performed during workflow execution, when tasks are scheduled by a workflow engine as "ready to be taken". For that purpose, two popular techniques "push" and "pull" are used, giving users the opportunity to choose tasks they will perform. The order of individual tasks is determined by a workflow model.

The third important area related to time representation and management is artificial intelligence, in particular *Temporal Constraint Networks* (TCN) [5]. When comparing time management in production workflows and temporal constraint networks the following similarities and differences could be identified. In both approaches, temporal information is modeled in the interval form (e.g. an event/task takes between 20 and 30 minutes). Both relative and absolute time are used as well as a set of temporal constraints. However, they are suitable for fundamentally different environments. Having a set of unary and binary temporal constraints specified, the TCN approach is concentrated, for example, around the following reasoning: finding all feasible times that a given event may occur, finding all possible relationships between two given events, and generating two or more scenarios consistent with the information provided [5]. On the other hand, in production workflows all expected scenarios are defined in advance by a workflow model including the duration of individual tasks and their possible relationships. Each scenario is called an instance type. The model also includes a set of various temporal constraints. During the modeling of the process it is necessary to verify whether these constraints are consistent with the definition of a workflow model, as well as being mutually consistent. Note also that the notion of temporal consistency used by [5] is different from the one used in our work, as we will illustrate later in the paper. On the other hand, during workflow execution the question is whether these constraints can be fulfilled in the instance being executed, and if not, the location of the problems. Furthermore, one task may belong to several instance types at the same time. Therefore, during workflow execution the same task will be executed a number of times, each time in different process instances and each time with different start and finish time. Hence, the possible start and finish time of a task can be calculated only

in the context of an instance and only when that instance starts. However, even when an instance is activated, that does not guarantee that a task is going to be executed at all, since a different execution path may be taken by the instance.

Other relevant research where problems of time management have been investigated include *temporal data bases* where the emphasis is on storage and retrieval of temporal data, and in *real-time software engineering* with the emphasis on specification of properties of real-time systems [15]. However, none of the above technologies are completely suitable for the time management of predefined processes such as production workflows.

3 Basic Workflow Concepts

The workflow model used in this paper is defined as a partially ordered set of workflow objects: *tasks, conditions* and *flows* [22, 24, 25]. A *task* represents the work to be done to achieve some given objectives. It can represent either automated or manual activity that is performed by assigned processing entities. A task can be initial, final or intermediate. A *decision* (also called *condition*) object is used to represent alternative paths in the workflow specification depending on a conditional value(s). Both tasks and conditions form a set of workflow graph *nodes*. Each node is identified by an unique identifier. A *flow* defines the connection between any two objects (other than flows) in the workflow. Each flow has one source and one sink node. To enhance the understanding of a workflow model, a graphical representation of its objects is used as shown in Fig. 1. Therefore, a

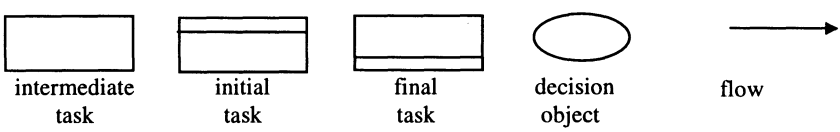

Fig. 1. Workflow modeling objects

workflow graph is a directed graph constructed from these objects according to specific syntax rules [25]. An example of a workflow model is represented in Fig. 2. After the initial task is executed, task 15 will be executed concurrently with one of the tasks 3, 4 or 5. The structure used to model the selection among more than one task is called an alternative structure. After the execution of task 4, either task 11 or task 12 will be executed. After task 5 has been executed, both tasks 6 and 7 will be executed in parallel. This structure is called a concurrent structure. Task 8 will wait until both 6 and 7 finish and then it will start executing. These three tasks form the structure called a synchronizer. Task 14 is executed after either task 3 or task 11. This is an executive "or-join" structure.

The terminal task 16 will start executing only after both tasks 15 and one of the tasks 8, 12 or 14 finish their execution. The introduced workflow structures will be described in more detail in the following section. In general, a workflow

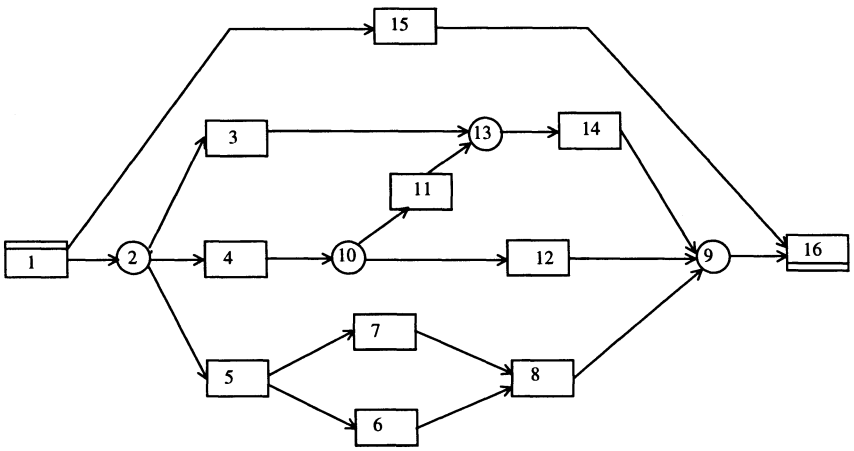

Fig. 2. An example of a simple workflow graph

model may have more than one terminal node. For these complex models we use an additional assumption that each process instance may finish at only one terminal node. All process instances that finish by one particular terminal node are modeled by one *execution sub-graph*. A model often consists of several execution sub-graphs, each corresponding to one terminal node. However, without loss of generality, in this paper we concentrate on a workflow model with only one terminal node, i.e. a model with one execution sub-graph.

Different alternative executives of a workflow, introduced by split nodes (decision nodes) are called *process instance types* or *instance types* for short. Obviously, a workflow model without decision nodes represents only one instance type. Finally a *workflow instance* is a single, individual instance of a process.

A workflow model can be simplified through the abstraction concept. In other words, it is possible to encapsulate a workflow model at one level of abstraction into a single task, called a *nested task*. As we do not consider repetitive structures at this stage, we also use a single nested task to encapsulate a loop concept.

It is important to point out that further on we assume syntactical correctness of a workflow model. That means that all decision nodes are exclusive (only one of the alternative paths can be selected) and complete (for all instances of a workflow, one of the alternatives will be always selected). Also, we assume that there is no deadlock and lack of synchronization. Some examples of the incorrect workflow structures are shown in Fig. 3. Both of them would provoke a deadlock

during a workflow execution. For more details on workflow modeling and verification issues see [22, 26, 28].

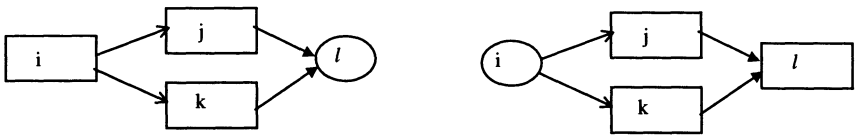

Fig. 3. Examples of incorrect workflow structures

4 Time Issues in Production Workflows

The purpose of this section is to introduce basic time notation and to set up the terminology such as relative and absolute time, temporal constraints and temporal consistency. These will then be used in the following sections for the formal modeling of a selected temporal constraint and verification of consistency.

4.1 Absolute and Relative Time

In production workflows we use two different aspects of time: *relative* and *absolute* time. In a workflow model, we operate with relative time, i.e. the expected duration of individual tasks and the duration of instance types are expressed as relative time values of a certain granularity, e.g. two hours, 10 minutes. On the other hand, during workflow execution we operate with absolute (or real) time. This means that each task in a workflow instance has its beginning and end time expressed as absolute time values of certain granularity, e.g. a task started at 2.00 p.m. and finished at 3.00 p.m. on 1 July.

4.2 Temporal Constraints

Temporal constraints are different rules that regulate the time component of a business process. From the business perspective they are defined by laws and regulations, business policies (or formal corporate rules), common practices, as well as mutual agreements and expectations related to efficiency/productivity of business practices. Temporal constraints are complex enough to be captured separately as an aspect of workflow modeling rather than covered by the workflow execution properties [10, 25].

We adopt the following classification of temporal constraints for production workflows. Temporal constraints are classified as *basic temporal constraints* (also

called *duration constraints), limited duration constraints, deadline constraints* and *interdependent temporal constraints.* Basic temporal constraints and deadline constraints are defined for the individual tasks, interdependent temporal constraints are defined for pairs of tasks while limited duration constraints are defined for one or more instance types.

Basic temporal constraint limits the expected duration of an individual task in a workflow model. The duration of the task is specified either precisely by using single relative time value (e.g. a task takes 1 hour), or as an interval of two relative time values (e.g. a task takes between 20 and 30 minutes) representing its expected minimum and maximum durations.

A limited duration constraint limits the duration of a process represented by a workflow model. For example: "A verification of each loan application should take no more than 8 hours". Note that in this case the limited duration constraint limits the duration of all instance types as well.

A deadline constraint, in terms of absolute time, limits when a task should start or end during workflow execution (e.g. the deadline for a grant application is 23 June). If a deadline constraint is defined for the final task of a workflow model with one terminal task, then it applies to all instance types of that workflow specification.

An interdependent temporal constraint limits the time distance between two tasks in a workflow model. The time distance is represented as a relative time value. For example: *a project proposal must be ready for review at least 7 days before the meeting and a report must be prepared two weeks after the meeting.*

In this paper, we limit our consideration to only the temporal constraints introduced above. In the following sections we will present their formal specifications.

4.3 Temporal Consistency

Closely related to temporal constraints is the problem of *temporal consistency.* In this paper, we propose the following definition. *A temporal constraint is consistent with a given workflow model if and only if it can be satisfied based on the syntax of the workflow model and the expected minimum and maximum durations of workflow tasks.*

Temporal consistency is of fundamental importance for workflow modeling since a workflow model may be correct in terms of control flow but may still have inconsistent temporal constraints. For example, suppose that in a workflow model represented in Fig. 2, a temporal constraint is defined such that task 12 should start 2 days after task 3. However, as tasks 3 and 12 always belong to different instance types the time distance between them cannot be meaningfully determined. Thus this temporal constraint is inconsistent with workflow model flow constraints, although the workflow model is correct in terms of its control flows.

Note that the notion of temporal consistency used in this paper is "stronger" than the notion of temporal consistency used in the literature dealing with temporal constraint satisfaction, see for example [5]. In the literature, a set of tem-

poral constraints is considered to be consistent if and only if there exists *an* assignment of values to the temporal variables involved that satisfies all the constraints. Here, a set of temporal constraints is consistent if and only if for *all* assignments of values to temporal variables involved (i.e. expected minimum and maximum duration of workflow tasks) and the given workflow model, all temporal constraints are satisfied.

5 Basic Temporal Constraint—A Duration Constraint

In our model, the *duration constraint* is a mandatory temporal constraint. It means that the duration for each task has to be specified. We assume a decision node has "0" duration, i.e. it is considered to be instantaneous. Also, at this stage, we do not consider *transfer time* between two nodes in a workflow graph. Therefore, without loss of generality, we assume that flows have no duration as well. So, we define the duration constraint in the following way:

Let N represent a set of identifiers of all workflow nodes. Suppose that each node has an unique identifier i. For every $i \in N$, we define duration of a node i, $d(i)$ as follows:

$$\forall i \in N \; d(i) = \begin{cases} [m(i), M(i)] & \text{if } i \text{ corresponds to a task,} \\ 0 & \text{if } i \text{ corresponds to a decision node.} \end{cases} \quad (1)$$

where $m(i)$ and $M(i)$ are two relative time values respectively representing the minimum and maximum duration of a task i.

It is obvious that for each task i: $m(i) \leq M(i)$. Also, in a case where $m(i) = M(i)$, we say that a task i has precise duration and we use only one value for its duration constraint.

To graphically model duration constraint, we introduce the concept of *the duration space* as follows:

$$D = \{(m, M) \in R^2 : m \leq M\} \quad (2)$$

where axes represent minimum and maximum duration m and M.

The duration of a given task is modeled as a point in duration space D with a label representing the identifier of that task as shown in Fig. 4. Note that all points situated on the diagonal have precise duration, by definitions 1 and 2.

Based on the introduced concept of a task in D, we will now analyze typical workflow structures and their durations.

5.1 Duration of a Sequential Structure

A sequential structure is the most basic construct of workflow modeling. It is shown in Fig. 5. Tasks i and j will be executed in sequential order, i.e. task j will begin executing only after task i has finished executing. To denote the order of individual tasks we use the notation: $i \cdot j$.

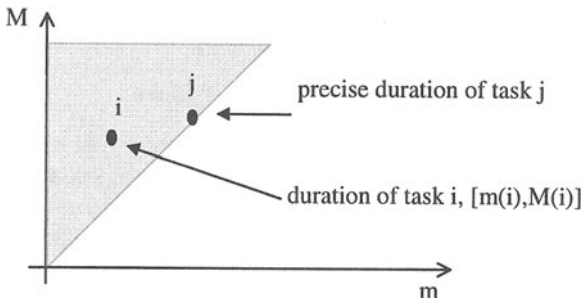

Fig. 4. Tasks in Duration space

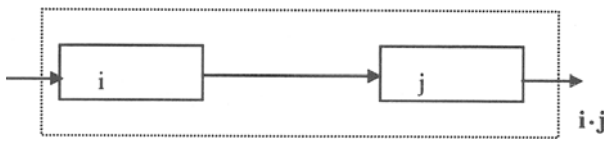

Fig. 5. Simple sequential structure

Based on the definition of a sequential structure, its duration can be calculated as follows:

$$d(i \cdot j) = [m(i), M(i)] + [m(j), M(j)] = [m(i) + m(j), M(i) + M(j)] \quad (3)$$

The resulting duration $d(i \cdot j)$ is represented visually in D as shown in the Fig. 6.

Note that, a sequential structure of two tasks with precise durations (e.g. k and l shown in Fig. 6), has precise duration too.

5.2 Duration of an Alternative Structure ("Or-Split")

This construct is used to select one of the several mutually exclusive alternative paths in a workflow. An example of an alternative structure with two alternative paths is shown in Fig. 7. After task i finishes its execution, based on a decision, either task k or task l will be selected for execution.

This alternative structure is represented in the duration space D as one of the two possible sequential structures: either as $i \cdot k$ or as $i \cdot l$.

5.3 Duration of an Exclusive "Or-Join" structure

As shown in Fig. 8, an exclusive "or-join" structure is used to represent the situation where only one of the incoming flows to decision node k will be executed. Before task l, either task i or task j will be executed.

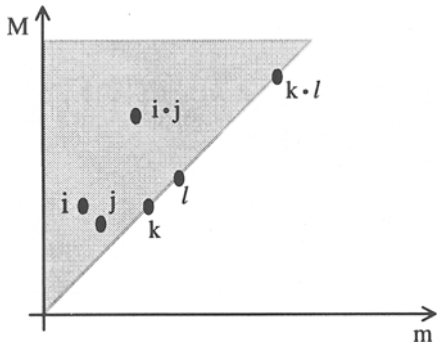

Fig. 6. Duration of sequential structures

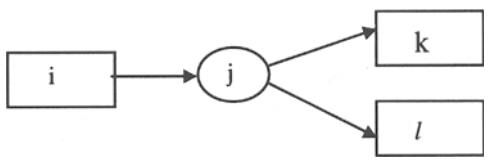

Fig. 7. Alternative ("Or–Split") structure

To present this structure in D we use one of the following two sequential structures: either $i \cdot l$ or $j \cdot l$.

5.4 Duration of Concurrent and Synchronized Structures

A concurrent (or "and-split") structure is used to present the parallel execution of two or more paths within a workflow. A simple concurrent structure is presented in Fig. 9. After completing task i, both tasks j and k will start executing simultaneously *independently* of each other.

If a task has two or more incoming flows we call that structure a *synchronizer*. A synchronizer with two incoming flows is shown in Fig. 10. Both tasks i and j will execute simultaneously independently of each other. Task k will wait for both tasks to complete their execution before it starts executing.

As we limit our consideration in this paper to workflow models with one final task, if a workflow model is correct, each synchronizer originates in one concurrent structure [25]. Therefore, for time consideration we combine a synchronizer with a concurrent structure as shown in Fig. 11. To represent that tasks j and k are executed in parallel, we use the notation $j \parallel k$.

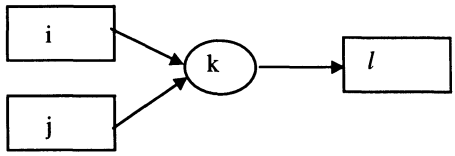

Fig. 8. Exclusive "or-join"

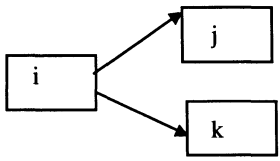

Fig. 9. A concurrent structure

To calculate the duration of $j \parallel k$ we use the following formula:

$$d(j \parallel k) = [\max(m(j), m(k)), \max(M(j), M(k))] \tag{4}$$

Hence, task l will wait until both tasks j and k are finished to start executing. If both j and k finish execution within the minimum period of time i.e. $m(j)$ and $m(k)$, a task l will wait for $\max(m(j), m(k))$. Similarly, if both tasks take the maximum time $M(j)$ and $M(k)$, a task l will wait for $\max(M(j), M(k))$. Various cases of tasks j and k as well as the resulting duration $d(j \parallel k)$ in D are presented in Fig. 12.

If both tasks j and k have a precise duration, their combination $(j \parallel k)$ has a precise duration as well (as represented in case (a)). In case (b), the resulting duration is determined by task k, while in case (c) it is determined by both tasks j and k.

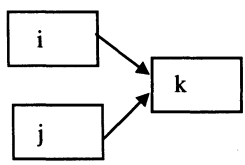

Fig. 10. A synchronizer

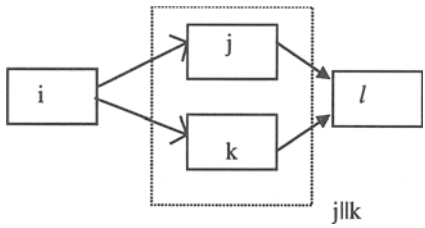

Fig. 11. A combination of a synchronizer with a concurrent structure

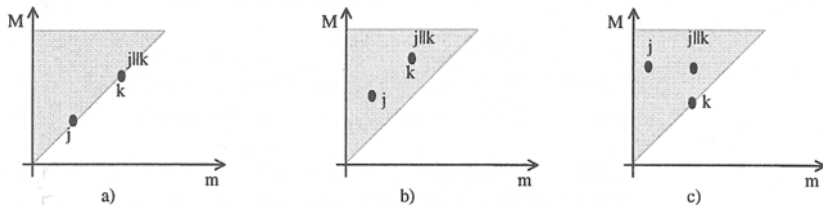

Fig. 12. Duration of a synchronizer structure

5.5 Duration of an Instance Type

An instance type is represented in a workflow graph W as a combination of sequential, concurrent and synchronizer structures. Based on the introduced rules for calculation of durations of these constructs, the duration of an instance type t can be calculated and represented as a natural extension of the concept of task duration, i.e. as an interval of its minimum and maximum duration:

$$d(t) = [m(t), M(t)] \tag{5}$$

Let a set T represents a set of all instance types for a given workflow W, let $\| T \| = n$

$$T = \{t_1, t_2, \ldots, t_n\} \tag{6}$$

such that

$$\forall k \; t_k = \{i_1, i_2, \ldots, i_m\} \tag{7}$$

where $\forall l \; i_l$ is a task in the workflow graph.

Note that the problem of enumerating all possible instance types represented by a workflow model is exponential in nature due to the potential exponential expansion generated by consecutive decision nodes.

Furthermore, we say that a task i is a *mandatory task* for workflow W if and only if:

$$\forall t_k \in T \; i \in t_k, k = 1, \ldots, n \tag{8}$$

Thus, a task i is mandatory if and only if it belongs to all instance types. In particular, the initial task and the terminal task (in a workflow model with one terminal task) are mandatory tasks for a workflow model.

A task j is an *optional task* for a workflow W, if and only if:

$$\exists k \in \{1, \ldots, n\}\ t_k \in T : j \notin t_k \tag{9}$$

These two definitions clearly indicate that not all tasks are executed by all workflow instance types. We will use these terms later in the paper, for verification of consistency of temporal constraints.

5.6 Duration of the Shortest/Longest Instance Types

An important question related to a workflow model is to find durations of the shortest/longest instance types.

An instance type t_l is the *longest instance type* in a workflow model if and only if:

$$\forall t_k \in T\ M(t_l) \geq M(t_k),\ k = 1, \ldots, n \tag{10}$$

An instance t_s is the *shortest instance type* in a workflow model if and only if:

$$\forall t_k \in T\ m(t_s) \leq m(t_k),\ k = 1, \ldots, n \tag{11}$$

Note that $m(t_l)$ and $M(t_s)$ represent respectively the minimum duration of the longest instance type and the maximum duration of the shortest instance type.

To calculate the shortest/longest duration instance type, we developed the *Duration algorithm.* The duration algorithm is a generalization of the shortest path partitioning algorithm, presented in [8]. The generalization is made in the following two aspects. First of all, the algorithm is specially designed for workflows rather than general graphs. It incorporates the semantic of workflow structures such as: concurrent, synchronizer, alternative and exclusive "or-join" structures. The other generalization is from a concept of a path in a graph into the concept of a workflow instance type. In workflows, we are not interested in the shortest path but rather in the duration of the shortest/longest workflow instance type which usually consists of several parallel paths (e.g. in an instance type, one path may split into several parallel paths which may merge again into one path). For example, an instance type of a workflow graph presented in Fig. 2, consists of the following tasks: 1, 4, 11, 14, 15 and 16.

Before the duration algorithm is presented, we shall introduce some notations:

- For calculation purposes, each node i (a task or a decision node) has a label in the form of pair $(\text{CumMin}(i), \text{CumMax}(i))$ which represents the cumulative minimum and maximum distance between the beginning of the initial task up to the end of a node i in an instance type. Calculation of the cumulative distance of other nodes is based on the semantic of workflow constructs.

- The initial task has the identifier 1. For the initial task: CumMin(1) = $m(1)$ and CumMax(1) = $M(1)$

The Duration Algorithm (for the shortest workflow instance type)

Input: A workflow graph with specified duration constraints for each task,
Output: Duration of the shortest workflow instance type,

Step 1: //initialization
Set the cumulative minimum and maximum of the initial task to
CumMin(1) = $m(1)$ and CumMax(1) = $M(1)$;
Set the cumulative minimum and maximum of all other nodes to 0
CumMin(i) = CumMax(i) = 0;
Create two collectively exhaustive lists of node identifiers:
NOW and NEXT
Initially NOW = {1} and NEXT = {}

Step 2: If NOW is not empty, take the first index from it,
say identifier u
otherwise go to Step 4

Step 3: Delete u from NOW
Find all successor nodes of u
For each successor node v
Find the number of predecessors npred of v
If npred = 1 then
calculate cumulative min and max of v as follows:
CumMin(v) = CumMin(u) + $m(v)$;
CumMax(v) = CumMax(u) + $M(v)$;
add index v to the list NOW
else //v has more than one predecessor
if v is a task then
// a synchronizer is detected
CumMax(v) = max((CumMax(u) + $M(v)$), CumMax(v))
CumMin(v) = max((CumMin(u) + $m(v)$), CumMin(v))
else //an exclusive "or-join" structure is detected
if CumMin(v) = CumMax(v) = 0 then
//decision node is visited for the first time
CumMax(v) = CumMax(u)
CumMin(v) = CumMin(u)
else //decision node is not visited for the first time
if CumMin(v) > CumMin(u) then
CumMin(v) = CumMin(u)
CumMax(v) = CumMax(u)
add v to NEXT if it is not already there
// therefore NEXT contains nodes

```
                // with more than one predecessor
        Return to Step 2

Step 4: if NEXT is empty then
            stop the algorithm
            cumulative minimum and maximum duration of the final node
            contain respectively, minimum and maximum duration of
            the shortest workflow instance type
        else
            from NEXT select all nodes ready for calculation
            transfer them to NOW
            // (a node form NEXT is ready if all its predecessors have
            // their cumulative minimum and maximum already calculated).
            Return to Step 2.
```

To trace the shortest instance type, it is necessary to follow flows from the final task backwards. In that process, if a synchronizer structure is detected it is necessary to follow backwards all incoming flows to that structure. If an alternative "or-join" structure is detected, from a decision node v select the node u with CumMin(u) = CumMin(v).

This algorithm can be easily modified to calculate the duration of the longest instance type. The code printed in *italics* font should be replaced with the following one:

```
else // an exclusive or-join structure is detected
    if (CumMax(v) < CumMax(u)) then
        CumMax(v) = CumMax(u)
        CumMin(v) = CumMin(u)
```

5.7 Complexity of the Algorithm

In the worst possible case a workflow model consists of concurrent, synchronizer, exclusive "or-join" and alternative structures only but does not contain any sequential structure. In this case the complexity of the duration algorithm is equal to the complexity of the shortest path partitioning algorithm. More precisely, the complexity is: nf where n represents a number of nodes and f number of flows in a workflow model. Therefore, the duration algorithm has a polynomial time complexity of $O(n^2)$.

The duration algorithm can also be generalized to calculate the shortest/longest instance type in a workflow model with more than one terminal node, i.e. more than one execution sub-graph. First of all, each execution graph needs to be determined by tracing back all instances from each terminal node. Then, for each execution sub-graph, the duration algorithm can be applied to calculate the

shortest/longest instance type for that particular sub-graph. The resulting shortest/longest instance type for the whole workflow will be the minimum/maximum of all shortest/longest instance types of all execution sub-graphs.

The presented duration algorithm has another interesting feature. If a workflow model consists only of decision nodes and tasks, but no synchronizers and parallel structures, the duration algorithm could be easily modified into the partitioning shortest path algorithm. On the other hand, if the workflow model consists of synchronizer and parallel structures only, but no alternative and exclusive "or-join" structures, then there would be only one workflow instance type. If each task has the earliest start and the latest finish time assigned, instead of minimum and maximum duration, then the duration algorithm for the longest workflow instance type can be easily modified into the CPM algorithm. However, it is necessary to stress again that the workflow model combines all four structures: parallel, synchronizer, alternative and exclusive "or-join" structure. This makes the model much more complex than a model used for the CPM algorithm (i.e. precedence diagram) or a graph used for the shortest path algorithm.

6 Other Temporal Constraints and Rules for Verification of Their Temporal Consistency

Based on the formalisms introduced in the previous sections, we will now present other temporal constraints that often restrict business processes.

6.1 Limited Duration Constraints for a Workflow Process

A limited duration constraint limits the duration of a process represented by a given workflow model. For example: Each loan application should be processed and a decision made within *no more* than two working days. To ensure the quality of the process performed, the processing of each loan application should take *at least* one day. Process duration is calculated as the distance between its initial and final tasks.

Formally, a limited duration constraint between the initial and final task is defined as a relation:

$$L \subseteq \{1\} \times \{n\} \times T_{\mathrm{L}} \times R \tag{12}$$

where:
$\{1\}$ and $\{n\}$ represent identifiers of the initial and final tasks, respectively
T_{L} represents a type of limited duration constraint $T_{\mathrm{L}} = \{\leq, \geq\}$ and
R represents a set of relative time values of a certain granularity, e.g. 1 day, 3 months, etc.

To represent the longest duration of a process, i.e. the longest time distance from task 1 to task n, we use the notation $M_{\mathrm{l}}(1\text{–}n)$. The following consistency rule applies:

Rule 1: *A limited duration constraints* $L(1, n, \leq, p)$ *which limits the process to take no more than p time is time consistent if and only if:*

$$M_1(1\text{–}n) \leq p \tag{13}$$

where $p \in R$ *is a relative time value which represents the duration limit and* $M_1(1\text{–}n)$ *is the longest duration of a process calculated from the initial task* 1 *to the final task n.*

Note that $M_1(1\text{–}n)$ also corresponds to the maximum duration of the longest instance type for a given workflow. Therefore, to calculate $M_1(1\text{–}n)$ we need to apply the duration algorithm.

This limited duration constraint can be represented as a triangular area in D as shown in Fig. 13. If this constraint is consistent, the resulting point representing the duration of the longest instance for a given workflow model will be located within the gray area.

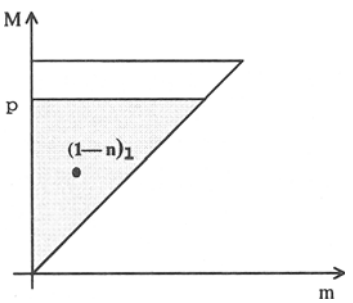

Fig. 13. Limited duration constraint – temporal consistency rule 1

Similarly, another type of limited duration is defined for the shortest duration of a process, i.e. the shortest time distance from the initial task 1 to the final task n. A following consistency rule can be formulated:

Rule 2: *A limited duration constraint* $L(1, n, \geq, p)$ *which limits the process to take at least p time is time consistent if and only if:*

$$m_S(1\text{–}n) \geq p \tag{14}$$

where $p \in R$ *is a relative time value which represents the duration limit and* $m_S(1\text{–}n)$ *is* the shortest duration of a process calculated from the initial task 1 to the final task n.

If the workflow model is consistent, then the resulting point representing duration of the shortest workflow instance type must be located within the gray area as shown by Fig. 14.

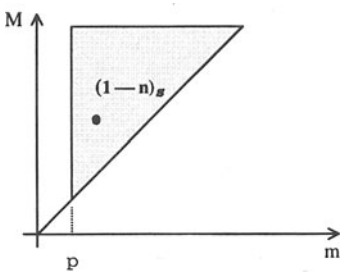

Fig. 14. Limited duration constraint: temporal consistency rule 2

Note that $m_S(1–n)$ corresponds to the minimum duration of the shortest instance type for a given workflow. Therefore, to calculate $m_S(1–n)$ we need to use the duration algorithm.

6.2 Deadline Constraints – Real Time Constraints

A deadline constraint defines when a task should start or finish in terms of absolute time. Formally, a deadline constraint D is represented as a relation

$$D \subseteq N_t \times T_D \times A \tag{15}$$

where: $N_t \subset N$ represents a set of workflow task identifiers (a subset of a set of all workflow node identifiers),
T_D represents a type of deadline constraint $T_D = \{b, e\}$, where b applies to the beginning and e to the end of a task, and
A represents a set of absolute time values of a certain adopted granularity.

Suppose that a deadline constraint is defined for a task i. If task i is a mandatory task for a workflow model, then the deadline constraint will apply to all instance types of that workflow model. On the other hand, if task i is an optional task for a workflow model, then during workflow execution task i may not be executed at all and, in this case, the deadline constraint will not be fulfilled. We call this type of deadline constraint a *conditional deadline*. Note that in both cases a deadline constraint will apply only to those instance types containing task i.

An obvious practical question related to a deadline constraint would be: Is it possible to finish a particular task i by the deadline $Date1$? In other words, is it possible to satisfy the deadline constraint $D(i, e, Date1)$? If task i is an optional task, this constraint represents a conditional deadline and therefore it may not be satisfied during workflow execution. However, even if task i is a mandatory task, we cannot predict the absolute time when task i will be executed during workflow execution. Therefore, this constraint can be verified only during workflow execution and only when the instance containing task i

starts executing. However, based on the expected durations of the workflow tasks, we can still provide some help to a workflow user. For example, we can determine the earliest/latest absolute time when an instance to which task i belongs should start, in order for the deadline to be met. For that purpose, we use the *duration algorithm* to calculate the longest/shortest time distance between the initial task and task i for all instances to which task i belongs. For the longest and the shortest time distance between the initial task and task i, we use the following notations respectively: $M_{\mathrm{l}}(1\text{–}i)$ and $m_{\mathrm{s}}(1\text{–}i)$. Therefore, to determine the earliest/latest absolute time when an instance containing task i should start, it is necessary to subtract the durations $M_{\mathrm{l}}(1\text{–}i)$ or $m_{\mathrm{s}}(1\text{–}i)$ from a deadline defined for task i. For example, suppose that task i should start on 15 October and the shortest and the longest distance between the initial task and task i are 8 and 10 days respectively. Hence, the earliest date when an instance containing task i should start is 5 October, while the latest date is 7 October.

The other type of verification is related to pairs of duration constraints and their mutual consistency. Suppose that two deadline constraints are given: $D(i, e, Date1)$ and $D(j, e, Date2)$ as shown in Fig. 15.

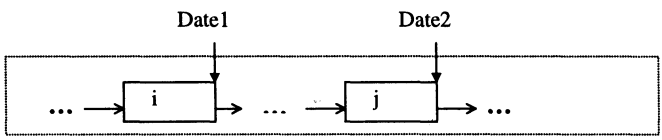

Fig. 15. A workflow model with two deadline constraints

Individual consistency of each deadline constraint can be verified only during workflow execution when finish times of both tasks i and j can be determined in terms of absolute time. At the same time, it is important to verify consistency of the distance between two absolute times before workflow execution in order to determine a potential problem.

Both $Date1$ and $Date2$ are absolute time values and as such, they cannot be represented in duration space D. (Recall that in D we model relative but not absolute time). However, the distance between two precise calendar dates $Date1$ and $Date2$ is a relative time value (e.g. 2 days) which can be represented in duration space D. We use the notation: $dist(Date1, Date2)$, where $dist$ is a function that returns the relative time value representing the difference in time between two absolute time values. This feature enables us to check mutual consistency of pairs of deadline constraints, i.e. the consistency of the distance between pairs of deadline temporal constraints in D in the following way:

(a) If tasks i and j do not belong to the same instance type, or they are concurrent tasks within the same instance type, the distance between them cannot be calculated or rather is undefined. Therefore, mutual consistency

of this pair of temporal constraints cannot be determined before workflow execution.

(b) If both tasks i and j belong to the same instance type, and they are not concurrent tasks, then it is possible, based on the expected durations of individual tasks, to calculate the distance from i to j. In this calculation it is necessary to take into account all instance types to which both tasks belong. Therefore, we calculate the maximum time distance between tasks i and j, $M_l(i–j)$ by applying the duration algorithm starting from task i, as presented in Fig. 16.

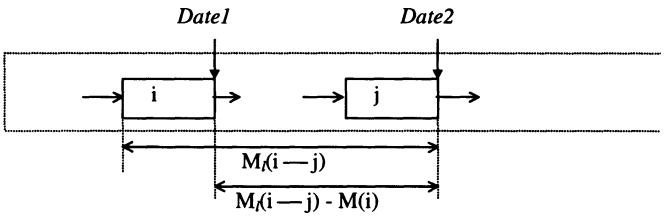

Fig. 16. The distance between two deadline constraints

The following consistency rule applies:

Rule 3: *A pair of deadline constraints* $D(i, e, Date1)$ *and* $D(j, e, Date2)$ *defined for two non-concurrent tasks* i *and* j *which both belong to at least one common instance type and task* i *is always executed before task* j*, is time consistent if and only if the following rule is satisfied:*

$$M_l(i–j) \leq dist(Date2, Date1) \tag{16}$$

where the function dist(Date2,Date1) returns the difference between two absolute times values $Date1$ *and* $Date2$ *and* $M_l(i–j)$ *represents the longest time distance between tasks* i *and* j *calculated for those instance types where both tasks* i *and task* j *belong to.*

Note also that by definition:

$$M(i–j) > M(i) \text{ and } m(i–j) > m(i) \tag{17}$$

This feature enables us to visualize this temporal constraint in D as shown in Fig. 17. The gray area limits the position of a task i in D.

The above reasoning can naturally be generalized for any number of deadlines.

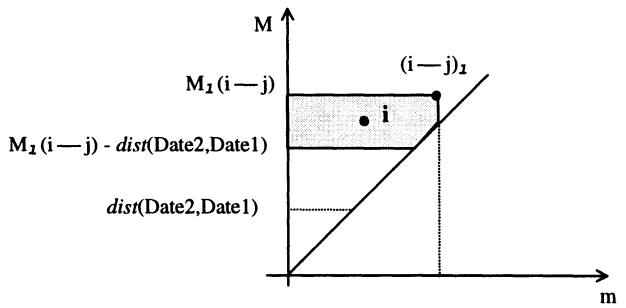

Fig. 17. Deadline temporal constraints in D

6.3 Interdependent Temporal Constraints

An interdependent temporal constraint limits the time distance between two tasks in a workflow. More precisely, it limits when a task should start/end relative to the beginning/end of another task in a workflow. If these two tasks are adjacent, i.e. there is a direct flow from one task to another, this constraint is called a *basic interdependent temporal constraint.* Since a direct flow exists between these two tasks, it follows that then both tasks always belong to the same instance type. However, the consistency of a basic interdependent temporal constraint can be verified only during workflow execution when the beginning and end time of both tasks are known.

Here we consider a more general case of interdependent temporal constraints defined for two non-adjacent tasks in a workflow model.

Non-adjacent interdependent constraint As we already pointed out, a non-adjacent interdependent temporal constraint limits the distance between two non-adjacent tasks is a workflow model. For example: task j should start *no later* than p time after task i finishes. Task j should start *no earlier* than p time after task i finishes. In both cases there is no direct flow between tasks i and j.

Formally this constraint is defined as a relation:

$$I \subseteq N_t \times N_t \times T_I \times R \tag{18}$$

where:
$N_t \subset N$ represents a set of workflow task identifiers (a subset of a set of all workflow node identifiers),
T_I represents a set of constraint types $T_I = \{\leq, \geq\}$, and
R represents a set of relative time values.

Note that if any of the tasks i and j is an optional task for a workflow model, then this task may not be executed at all during workflow execution. In this case,

an interdependent temporal constraint defined between these two tasks will not be satisfied.

In order to verify the consistency of an interdependent non-adjacent temporal constraint we need to consider the following two cases:

(a) If tasks i and j do not belong to the same instance type, then an interdependent temporal constraint which limits the distance from i to j is *inconsistent* as the distance between those two tasks cannot be calculated.

(b) Suppose that tasks i and j belong to the at least one common instance type and they are not concurrent tasks. To verify the consistency of a temporal constraint: $I(i, j, \leq, p)$ where p represents the duration limit, we need to apply the *duration algorithm*, to calculate the longest time distance from i to j, $M_1(i\text{–}j)$. Note that the resulting duration includes duration of both tasks i and j as shown in Fig. 18.

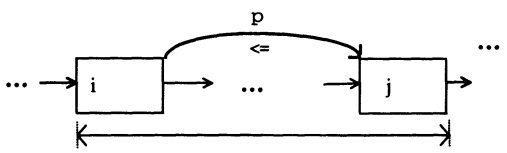

Fig. 18. Non-adjacent interdependent temporal constraint

The following consistency rule applies:

Rule 4: *Let tasks i and j be two non-concurrent tasks which belong to at least one common instance type and task i is executed before a task j. An interdependent temporal constraint $I(i, j, \leq, p)$ is consistent only and only if:*

$$M_1(i\text{–}j) - (M(i) + M(j)) \leq p \tag{19}$$

where $p \in R$ represents the duration limit and $M_1(i\text{–}j)$ is the longest duration from the beginning of a task i to the end of task j calculated for all instance types where both tasks i and j belong to.

Furthermore, it is obvious that for the consistent interdependent temporal constraint:

$$M_1(i\text{–}j) > p \tag{20}$$

Hence, this interdependent temporal constraint can be visualized in D as shown in Fig. 19.

Note that if at least one decision node exists between tasks i and j, some instance types will contain both tasks, while some other instance types will contain task i but not task j. An interdependent temporal constraint which

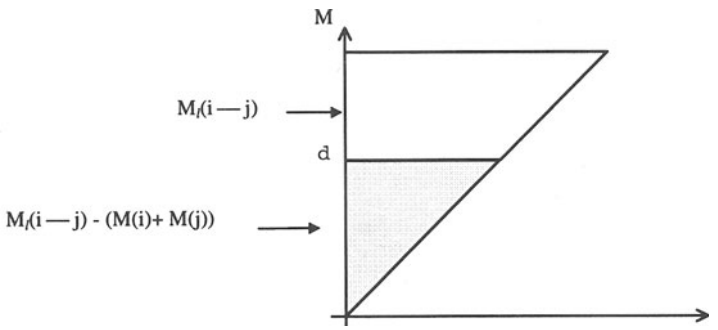

Fig. 19. Interdependent temporal constraint in duration space

limits the distance from task i to task j, will apply only to those instance types containing both tasks i and j.

A similar analysis applies to another case of interdependent temporal constraint as illustrated by the following consistency rule:

Rule 5: *Let tasks j and k be two non-concurrent tasks which belong to at least one common instance type and task j is executed before a task k. An interdependent temporal constraint $I(j, k, \geq, p)$ is consistent only and only if:*

$$m_S(j\text{–}k) - (M(j) + M(k)) \geq p \tag{21}$$

where $p \in R$ represents the duration limit and $m_S(j\text{—}k)$ is the shortest duration from the beginning of a task j to the end of task k calculated for all instance types to which both tasks j and k belong to.

It is very important to point out the following limitation which applies to the interdependent temporal constraints in production workflows. Suppose that two interdependent constraints are defined for tasks i, j and k, $I(i, j, \leq, p1)$ and $I(j, k, \leq, p2)$ as shown in Fig. 20. Those constraints cannot be replaced by a constraint $I(i, k, \leq, p1 + p2 + M(j))$, even in the case when all three tasks i, j and k belong to the same instance type. The reason for this is the following. By replacing those two constraints by one, a new temporal constraint that a task k should start *no later* than $(p1 + p2 + M(j))$ time after a task i, will be "artificially" imposed while the original two constraints will be lost and a task j will be unaffected by the new temporal constraint. In most business processes, this is not acceptable because one or both "lost" constraints could be "hard" temporal constraint imposed by the law.

All other cases of non-adjacent interdependent temporal constraints (e.g. a task j should start no later that p time after a task i starts or a task j should finish no later than p time after a task i finishes) can be analyzed in the

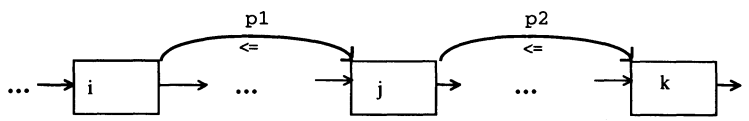

Fig. 20. An example of interdependent temporal constraints

similar way based on the previous two cases. Having that in mind, a limited duration temporal constraint can be treated as a special case of non-adjacent interdependent temporal constraints since the initial and the final tasks always belong to the same instance type.

7 Verification Algorithm

The above rules for the verification of temporal consistency of temporal constraints can be combined into the *Verification algorithm.* The algorithm detects all inconsistent and temporal constraints in a workflow model. The duration algorithm is used for the calculation of the shortest/longest instance types as well as minimum/maximum distance between any two tasks which belong to the same instance type(s).

The Verification Algorithm

Input: A workflow graph with specified duration, limited duration, deadline and interdependent temporal constraints
Output: A report containing a list of all inconsistent and potentially inconsistent temporal constraints

For each workflow node with an identifier i check its type
  if i is a task then
    if $M(i) < m(i)$ then
      report an error: "Basic temporal constraint for task i is not properly defined"

Identify all interdependent temporal constraints
If none defined then
  report: "No interdependent temporal constraints defined for this workflow model"
else for each interdependent temporal constraint
  identify its "from" node i and its "to" node j
  if i and/or j are not task identifiers then
    report an error: "The interdependent temporal constraint

```
        from i to j is not properly defined"
    else identify the duration limit p and its type
        //e.g. ≤ or ≥
        if p <= 0 then
            report an error: "duration limit not properly defined"
        else if a direct flow exits from i to j then
            the basic interdependent constraint is identified
            create an artificial task with minimum and maximum
            duration equal to the duration limit p
            //this will be used when applying
            //the duration algorithm
            else //it is a non-adjacent interdependent constraint
                apply the DetectInconsistency algorithm from i to j
                if the inconsistent case is detected then
                    report: "The interdependent temporal constraints
                    defined from i to j is inconsistent—tasks do not belong
                    to the same instance type or they are concurrent"
                else //the inconsistent case is not detected
                    if the constraint type is ≤ then
                        apply the duration algorithm to calculate M_l(i–j)
                        if M_l(i–j) – (M(i) + M(j)) > p then
                            report: "The interdependent temporal constraint
                                defined from i to j is inconsistent:
                                the duration limit p is too short"
                    else // the constraint type is ≥
                        apply the duration algorithm to calculate m_s(i–j)
                        if m_s(i–j) – (m(i) + m(j)) < p then
                            report an error: "The interdependent
                                temporal constraint defined from i to j
                                is inconsistent: duration limit p
                                is too long"
Identify if any limited duration constraint is defined
If none defined then
    report "No limited duration constraints defined"
else determine the type of constraint (≤, ≥)
    and the duration limit p
    if the type is ≤ then
        apply the duration algorithm to calculate M_l(1–n)
        if M_l(1–n) > p then
            report an error: "the limited duration constraint is inconsistent"
    else //the type is ≥
        apply the duration algorithm to calculate m_s(1–n)
        if m_s(i–j) < p then
            report an error: "The limited duration constraint
            is inconsistent"
```

```
Identify if any deadline constraints is defined
If none is defined then
    report: "No deadline constraints defined"
else
    for each deadline constraint identify the node i for which is defined
        if i is not a task identifier then
            report an error: "Deadline constraint not properly defined
                node i is not a task"
        else //i is a task identifier
            if there is more than one deadline constraint defined then
                for each deadline constraint identify
                    the absolute time value representing the deadline
                    for that temporal constraint
                sort all absolute time values in the ascending
                    order: Date1, Date2, Date3,...
                form the pairs of adjacent deadline constraints
                    //e.g. (i, e, Date1) and (j, e, Date2);
                    //(j, e, Date2) and (k, e, Date3) etc.
                for each pair defined for tasks i and j
                    with the respective deadlines Datei and Datej
                apply the DetectInconsistency algorithm from i and j
                If tasks i and j don't belong to the same instance
                    or they are concurrent then
                    // case a)
                    report: "Mutual consistency of deadlines Datei and
                        Datej can be verified during workflow execution
                else //case b)
                    use the duration algorithm to calculate their
                        maximum distance M1(i—j)
                    calculate the distance dist(Datei, Datej)
                    if M1(i—j) − M(i) > dist(Datei, Datej) then
                        report an error: "Deadline temporal constraints
                            for tasks i and j are mutually inconsistent"
Stop the algorithm.
```

The presented algorithm for the verification of temporal constraints is based on the duration algorithm and also has a polynomial complexity.

To detect an inconsistent case, when two tasks are concurrent or do not belong to the same instance type, an algorithm called *DetectInconsistency* is used. To improve the readability and clarity of the *DetectInconsistency* algorithm, we use the *Scan* algorithm which checks if node j can be reached from node i in a given workflow model.

For the purpose of the Scan algorithm, for each node (a decision node or a task) in a workflow model we define a temporary label that has one of the three values:

"u" — unscanned

"s" — scanned
"r" — reached but not yet scanned

The Scan Algorithm

Input: Identifiers of two nodes i and j
// a node can be a task or a decision node
Output: A report containing one of the two options:
—a node j cannot be reached from a node i or
—a node j can be reached from a node i

Step 1: Suppose that v and $NODE$ represent node identifiers.
Initially $v = 0$ and $NODE = i$
Initially, all nodes have the label "u"

Step 2: Find all flows originating in $NODE$
//this is the process of scanning
For each node v at the head of each of those flows
if its existing label is "u" then
change the label to "r"
Update the label of $NODE$ to "s"

Step 3: If node j has a label "r" then
report that node j can be reached from node i
Stop the algorithm

Step 4: From a set of all nodes with label "r"
select one and call it $NODE$
Return to Step 2.
If there are no nodes with label r then
report that node j cannot be reached from node i
Stop the algorithm.

The Scan algorithm can be easily generalized to check if any of the nodes from a group of nodes can be reached from a single node i. In that case the algorithm will stop when any node from the group is reached or when the final task in a workflow model is reached.

To detect if two tasks belong to the same instance type and/or if they are concurrent, we use the *DetectInconsistency* algorithm. It is based on the following assumptions: If one task can be reached from another task, then these tasks belong to the same instance type and they are not concurrent. If two tasks i and j are concurrent, there is a common task (a part of a synchronizer structure) that can be reached from both tasks i and j (For example, in a workflow model represented in Fig. 2, task 8 can be reached from both tasks 6 and 7). Finally, if one task cannot be reached from the other and if these tasks are not concurrent, then they don't belong to the same instance type.

The DetectInconsistency Algorithm

Input: Identifiers of two tasks i and j
// supplied in that order by the Verification algorithm
Output: A report which contain one of the following possibilities
—tasks do not belong to the same instance type or
—tasks are concurrent (belong to the same instance type) or
—tasks belong to the same instance type and they are not concurrent

Step 1: Suppose that v represents a task identifier,
initially $v = 0$.
Suppose that control list CL is a list of task identifiers;
CL is initially empty.

Step 2: Apply the Scan algorithm from task i to task j
if task j cannot be reached from task i then
// check if tasks are concurrent
detect all tasks with label "s" with more than one incoming flows
// this is detection of synchronizer structures
for each of those tasks v
if all incoming flows to v, have all source nodes
with label "s" then
tasks i and j are not concurrent and
they don't belong to the same instance type
//because task j cannot be reached from task i
else // at least one incoming flow to v has label "u"
//there is a possibility that tasks i and j are concurrent
put v in the control list CL
apply the Scan algorithm to check if any task from CL
can be reached from j
if none can be reached from j then
tasks i and j are not concurrent and
they don't belong to the same instance type
else // the same task can be reached from both i and j
tasks i and j are concurrent and
they belong to the same instance type
else // task j can be reached from task i
tasks i and j belong to the same instance type and
they are not concurrent
Stop the algorithm.

8 An Example of a Production Workflow and Its Temporal Constraints

The specification and verification of temporal constraints in a production workflow will be illustrated by a simplified example of a workflow for "Postgraduate Application Processing" as represented in Fig. 21. The complete example is presented in [25].

The admission process starts when the admission office receives a postgraduate application. For each individual application, a new workflow instance is created. The first task is to verify the completeness and correctness of the application. A complete and correct application is sent to the respective department for review. After application processing is completed by the department, the application is forwarded to the respective dean. After the dean reviews the application, the application is sent back to the admission office for final processing. Note that if the application is initially incomplete or incorrect, there will not be any departmental and dean review. In this case the application is sent directly to the administrative officer for final processing. If the application is accepted, it is forwarded to the enrolment section for further processing. Three different tasks are then initiated concurrently. An enrolment letter is sent to the applicant, the applicant's details are put on the enrollment mailing list, and the department is notified again. After all three tasks are finished, the enrollment officer signs the report, and forwards the application to the central registry. If the application is rejected, the rejection letter is sent to the applicant and the application is forwarded to the central registry. Both successful and unsuccessful applications are filed in the central registry.

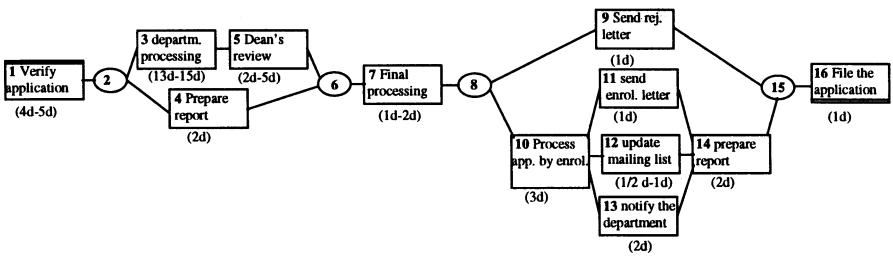

Fig. 21. Processing of postgraduate applications (a simplified example)

Suppose that the following temporal constraints are defined:

c1: Processing of a postgraduate application (from its initial to the final stage) should not take more than 30 working days (6 weeks).

c2: The deadline for the enrolment section to send applications to the central registry is 15 October.

c3: The department should receive the application 1 week after its submission.
c4 & c5: The department should be notified from the enrolment section no earlier than 10 days but no later than 15 days after it forwards the application to the dean.
c6: The enrolment officer should receive the successful applications by 10 October

Duration constraints for individual tasks are defined as presented on a workflow model. Note that the adopted granularity for the relative time is a day (d). For example, as shown in Fig. 21, task 1 takes between 4 and 5 days to execute and its duration consistency is represented as (4d–5d).

After the execution of the verification algorithm the following result will be obtained:

- The limited duration constraint **c1** is inconsistent—the duration of the longest duration of the process is 35 days. Possible improvements are to reduce duration (improve precision) of tasks 1, 3, 5 and 7.
- The interdependent temporal constraint **c3** is consistent with the specification of a workflow model. However, the existence of a decision node between tasks 1 and 3 implies that in some cases task 3 will not be executed at all.
- Two interdependent temporal constraints **c4** and **c5** are consistent—because the shortest distance between the end of task 3 and the end of task 13 is 10 days and the longest distance is 14 days.
- The deadline constraints **c2** and **c6** are mutually inconsistent because the time distance between the beginning of task 10 and the end of task 14 is 7 days, while the distance between two absolute deadlines 10 October and 15 October is 5 days. Therefore, if the first deadline is fulfilled, the second one is impossible to fulfill.

The same temporal reasoning can be applied to find answers on some specific questions. For example:

—What will happen if the duration of task 3: "Departmental processing" is increased/decreased?

To investigate the possible effects of the changed duration of task 3, the verification algorithm will be applied again and the consistency of all constraints will be reassessed. For example, if the maximum duration of task 3 is decreased for 5 days, the temporal constraint *c1* will become consistent.

—When should the department receive the application (the latest/earliest possible absolute time for the deadline defined by the constraint c6 *to be met?*

The duration algorithm will be used to calculate the shortest/longest distance from the beginning of task 3 to the beginning of task 10. Therefore, the department should receive the application 16–22 working days prior to 10 October. Based on a university calendar for a specific year (including public holidays)

the earliest and the latest calendar dates can be calculated.

—*What will happen if task 7 "Final processing by administrative officer" is removed from the workflow model?*

Temporal constraints *c1* and *c4* will be affected. It is necessary to apply the verification algorithm again to assess the effect this change of model may have to their consistency.

It is necessary to point out once again that this example represents only a simplified version of the actual workflow model. In the real model, many more temporal constraints are specified and it is a very time-consuming and error-prone process to verify them manually one by one, as well as to assess all the possible effects that a change in a workflow model may have on all temporal constraints. Therefore, we stongly believe that the help provided by our temporal model and algorithms is necessary for proper time management in production workflows.

9 Conclusion

Time is a fundamental concept of all workflow processes. As time cannot be preserved it must be prioritized, organized and modeled. We argue that currently there is no adequate formalism for time modeling in production workflows. The paper has introduced a complete framework for time modeling in production workflows, including specification and verification of a limited number of temporal constraints. The *Duration Algorithm* which calculates the duration of the shortest/longest workflow instance is also presented. The algorithm has an interesting feature: it is a workflow-specific generalization of two different categories of algorithms, the shortest path partitioning algorithm and the Critical Path Method (CPM). Based on the duration algorithm, the verification algorithm is developed with the purpose of verifying the consistency of temporal constraints.

We argue that the proposed temporal model is useful, generic, simple, yet very powerful. In addition to its simplicity, its main advantage is the possibility of the visualization of various temporal constraints. Better time management will lead to better coordination of tasks and/or business processes, better planning and scheduling in production workflows and possibly better understanding of workflow evolution.

Acknowledgments

This work was supported in part by the research grant 98/UQNSG048G. The authors would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

1. P. Barthelmess, J. Wainer. Workflow systems: A few definitions and a few suggestions. In: *Proc. COOCS'95*, Milpitas, CA, USA, 1995.
2. F. Burger, G. Quirchmayr, S. Reich, A. M. Tjoa. Using HyTime for modeling publishing workflows, *SIGOIS Bulletin,* **16**(1),1995.
3. Butler Report. *Workflow: Integrating the Enterprise.* The Butler Group, 1996.
4. P. Dadam et al. ADEPT—Next Generation Workflow Management System, ADEPT Project Department, DBIS, Germany, 1998.
5. R. Dechter, I. Meiri, J. Pearl. Temporal constraint networks, *Artificial Intelligence* **49**, 61–95, 1991.
6. E. Egger, I. Wagner. Negotiating temporal orders: The case of collaborative time management in a surgery clinic, *Computer Supported Cooperative Work* **1**, 255–275, 1993.
7. C. Ellis, G. Nut. Workflow: The process spectrum. In: *Proc. NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions*, Athens, Georgia, USA, 1996.
8. J. Evans, M. Edward. *Optimization Algorithms for Networks and Graphs*, M. Dekker: New York, 1992.
9. J. J. Feather, K. F. Cross. Workflow analysis: Just-in-time techniques simplify administrative process in paperwork operation, *Industrial Engineering* **20**, 32–37, 1988.
10. A. Geppert, P. Santanu, J. Eder, K. Doyle. Panel1: Workflow research and workflow products—Anything in common? In: *Proc. International Conference CAiSE'98*, Pisa, Italy, 1998.
11. D. Georgakopoulos, M. Hornick. An overview of workflow management: From process modeling to workflow automation infrastructure, *Distributed and Parallel Databases* **3**, 119–153, 1995.
12. I. Haimowitz, J. Farley, G. S. Fields, J. Stillman, B. Vivier. Temporal reasoning for automated workflow in health care enterprises. In: M. Adam, Y. Yesha (eds.), *Electronic Commerce: Current Research Issues and Applications.* Lecture Notes in Computer Science 1028, 1996.
13. J. G. Kobelius. *Workflow Strategies*, IDG Books, 1997.
14. D. E. Mahling, C. Woo, R. Blumenthal, H. Schlichter, T. Horstman. Workflow = OIS? A report of a workshop at the CSCW 94 conference, *SIGOIS Bulletin* **16**(1), 1995.
15. R. Maiocchi, B. Pernici. Time management systems, *IEEE Transactions on Knowledge and Data Engineering* **3**(4), 1992.
16. O. Marjanovic, M. E. Orlowska. Time modeling in workflows, *Technical report no. 413*, School of Information Technology, University of Queensland, Australia, 1997.
17. R. Marshak. Requirements for workflow products. In: D. Coleman (ed.), *Proc. Conference: GroupWare '92*, Morgan Kaufmann Publishers: San Mateo, California, 1992.
18. R. Medina-Mora, T. Winograd, R. Flores, F. Flores. The action workflow approach to workflow management technology. In: *Proc. CSCW'92 Conference*, 1992.
19. C. Mohan. Tutorial: State of the art in workflow management systems research and products. In: *Proc. 5th International Conference on Database Systems for Advanced Applications (DASFAA'97)*, Melbourne, Australia, 1997.
20. C. Mohan, G. Alonso, R. Gunthor, M. Kamath. Exotica: A research perspective on workflow management systems, *Data Engineering*, **18**(1), 1995.

21. J. M. Nicolas. *Managing Business and Engineering Projects*, Prentice Hall: New Jersey, 1990.
22. M. E. Orlowska, J. Rajapakse. On specification of task dependencies in transactional workflows. In: *Proc. International Conference on Systems Research, Informatics & Cybernetics. InterSymo'96*, 1996, pp.46–50.
23. Project Management Institute Standards Committee: *A Guide to the Project Management Body of Knowledge.* (http://www.pmi.org/pmi/publictn/pmbok/), 1996.
24. J. Rajapakse, M. E. Orlowska. Towards a graphical transactional workflow specification language. In: *Proc. Australian Systems Conference*, September, 1995.
25. W. Sadiq, M. E. Orlowska. Applying a generic workflow modeling technique to document workflows. In: *Proc. Second Australian Document Computing Symposium (ADCS'97)*, Melbourne, Australia, 1997.
26. W. Sadiq, M. E. Orlowska. On correctness issues in conceptual modeling of workflows. In: *Proc. 5th European Conference on Information Systems (ECIS'97)*, Cork, Ireland, 1997.
27. R. S. Sutton, A. G. Barto. Chapter 11: Case Studies—Job Shop Scheduling. In: *Reinforcement Learning: An Introduction*, MIT Press: Cambridge, MA, USA, 1998.
28. A. H. M. ter Hofstede, M. E. Orlowska, J. Rajapakse. Verification problems in conceptual workflow specifications. In: *Proc. 15th International Conference on Conceptual Modeling ER'96*, Cottbus Germany, 1996.
29. H. Wild, C. Darrouzet. Rhythms of collaboration, *Communication of the ACM*, September **38**(9),1995.
30. D. Worah, A. Sheth. What do advanced transaction models have to offer for workflows? In: *Proc. International Workshop on Advanced Transaction Models and Architectures*, Goa, India, 1996.
31. Workflow Management Coalition: *The Workflow Management Coalition Specifications—Terminology and Glossary*, 1996.

Olivera Marjanovic (BSc University of Sarajevo, MSc University of Belgrade) is currently employed as a Lecturer in the Information Environments Program at the Department of Computer Science and Electrical Engineering, University of Queensland. She is also associated with the Cooperative Research Centre for Distributed Systems Technology (DSTC) in Australia, working in the area of workflow technology. She has over nine years of professional experience in the area of computer science/information systems. Her current research interests inlude: workflow technology, intelligent agents, decision support systems and analysis and design of various information systems.

Maria E. Orlowska is Professor of Information Systems at the University of Queensland. Professor Orlowska's contributions to the database field have been demonstrated by her election for a period of five years as currently the only Australian representative on the prestigious international Very Large Databases Endowment (VLDB) - based in the USA. Since 1992, she has been associated with the Cooperative Research Centre for Distributed Systems Technology (DSTC) in Australia as the Leader of the Distributed Databases Unit. She has published over 120 research papers in international journals and conference proceedings. Professor Orlowska has expertise in: the theory of relational databases, distributed databases, workflows technology, various aspects of information systems design methodologies (including distributed systems), enhancement of semantic data modelling techniques by rigorous factors, transaction processing in distributed systems, concurrency control, and distributed and federated database systems.