published in: Layne Fischer (ed): Workflow Handbook 2001, Future Strategies INC. in association with Workflow Management Coalition 2000, ISBN 0-9703509-0-2, pp 109-132

Managing Time in Workflow Systems

Johann Eder
Department of Informatics Systems
University of Klagenfurt
A-9020 Klagenfurt, Austria
eder@isys.uni-klu.ac.at

Euthimios Panagos
AT&T Labs -- Research
180 Park Avenue
Florham Park, NJ 07932
thimios@research.att.com

Abstract

Even though currently available workflow management systems (WFMSs) offer sophisticated modeling tools for specifying and analyzing workflow processes, their time management support is rudimentary. Existing time management functionality mainly addresses process simulations (to identify process bottlenecks, analyze execution durations, etc.), assignment of deadlines to activities, and triggering of process-specific exception-handling activities (referred to as escalations) when deadlines are missed during process execution. In this paper, we address the crucial role of time management in the lifecycle of workflow processes. In particular, we describe how structural (i.e. execution order dependent) and explicit (i.e. fixed-date, periodic, upper- and lower-bound) time constraints can be modeled during process definition, validated during modeling and instantiation-times and, finally, monitored and managed during execution time.

1 Introduction

Today, the most critical need in companies striving to become more competitive is the ability to control the flow of information and work throughout the enterprise in a timely manner. Consequently, time-related restrictions, such as bounded execution durations and absolute deadlines, are often associated with process activities and sub-processes. However, arbitrary time constraints and unexpected delays could lead to time violations. Typically, time violations increase the cost of business processes because they require some type of exception handling [28]. Therefore, the comprehensive treatment of time and time constraints is crucial in designing and managing business processes. For instance, process managers need tools to help them anticipate time problems, pro-actively avoid time constraints violations, and make decisions about the relative process priorities and timing constraints when significant or unexpected delays occur.

Workflow management systems (WFMSs) improve business processes by automating tasks, getting the right information to the right place for a specific job function, and integrating information in the enterprise [14,16,22,34]. Although currently available commercial workflow products offer sophisticated modeling tools for specifying and analyzing workflow processes, their time management functionality is rudimentary [19,30]. In particular, existing time management functionality supports process simulations (to identify potential bottlenecks, analyze activity execution durations, etc.), assignment of deadlines to activities, and triggering of escalations (i.e. process-specific exception-handling activities) when deadlines are not met [3,4,13,18,23,32,33]. However, the consistency of these deadlines and the side effects of missing some of them are neglected.

It is imperative that current and future WFMSs provide the necessary information about a process, its time restrictions, and its actual time requirements to process modelers and managers.

- At build-time, when workflow schemas are defined and developed, workflow modelers need means to represent time-related aspects of business processes (activity durations, time constraints between activities, etc.) and check their feasibility;
- At run-time, when workflow processes are instantiated and their executions are started, process managers should be able to adjust time plans (e.g. extend deadlines) according to time constraints and any unexpected delays;
- During process execution, pro-active mechanisms are needed for notifying process managers about potential time constraint violations so that they can take the necessary steps to avoid time failures;
- If a time constraint is violated, the WFMS system should be able to trigger exception handling to regain a consistent state of the workflow instance;
- Workflow participants need information about urgencies of the tasks assigned to them to manage their personal work lists in accordance with the overall goals;

- Business process re-engineers need information about the actual time consumption of workflow executions to improve business processes;
- Finally, controllers and quality managers need information about activity start times and execution durations.

The latter two aspects are usually provided by workflow systems via workflow documentation (also referred to as *workflow history* or *workflow logging*) and monitoring interfaces. In this paper, we are mainly interested in the first three aspects. In particular, we address the following issues.

- Modeling of time and time constraints to capture the available time information;
- Pro-active time calculations to detect time constraint violations and raise alerts in case of potential future time violations;
- Time monitoring, deadline checking, and handling of time errors at run-time.

We should note, however, that the effectiveness of time management depends on the workflow kind, how detailed its description is, and whether there are external causes for time relevant events. For highly structured, production-based workflows, time requirements can be calculated with a high degree of accuracy. For administrative workflows, which span different organizations, depend on external events (e.g. waiting for a customer to reply), or may change their schema during execution (dynamic and ad-hoc workflows), time calculations are imprecise. Nevertheless, time planning, management, and controlling has to be done, and, to our experience, it is a common practice. Typically, time planning relies on estimates based on experience. Time management during the execution of a process becomes even more important in such environments, where time monitoring is essential for adjusting plans to avoid deadline violations. Therefore, any knowledge about time issues should be modeled and used during workflow execution.

The remainder of the paper is structured as follows. Section 2 describes the workflow model we assume in this paper, addresses activity durations and deadlines, and defines explicit time constraints. Section 3 presents how time information and time constraints can be modeled. Section 4 discusses time constraint satisfiability and monitoring during the lifetime of a workflow process. Section 5 outlines possible solutions to time violation that may occur at run-time. Section 6 touches upon schedule-based workflow executions. Section 7 offers a comparison with related work and, finally, Section 8 concludes our presentation.

2 Workflow Time Constraints

In this section, we begin by describing the assumptions we make about the workflow model that is used in the remainder of this paper. Then, we discuss the various time constraints that can be associated with processes and their activities.

2.1 Workflow Model

A workflow is a collection of *activities*, *agents*, and *dependencies* between activities. Activities correspond to individual steps in a business process, and agents (software systems or humans) are responsible for the execution of activities. Dependencies determine the execution sequence of activities and the data flow between them. Activities can be executed sequentially, repeatedly in a loop, or in parallel. Parallel executions can be *unconditional* (all activities are executed), *conditional* (only activities that satisfy a certain condition are executed), or *alternative* (any activity among several alternatives can be executed). In addition, workflows may contain *optional* activities. These activities may not be executed during a specific workflow instance in order to satisfy the time constraints associated with this instance.

Typically, workflows are represented by *workflow graphs* or *process maps*, where nodes correspond to activities and edges correspond to dependencies between activities. Figure 1 shows an example workflow graph that is based on the aforementioned notions. Activity *A* is the start activity. *A* is followed by a *conditional or-split*, and either *B* or *C* is executed next. If *C* is executed after *A*, an *and-split* follows *C*, and *E* and *F* are executed in parallel when *C* is completed. When both *E* and *F* complete their execution, *G* starts its execution. Activities *D* and *I* are optional. After *I*, there is an *alternative-split*, either *J* or *K* is executed (both are valid choices). Finally, *L* corresponds to the final activity of the workflow.

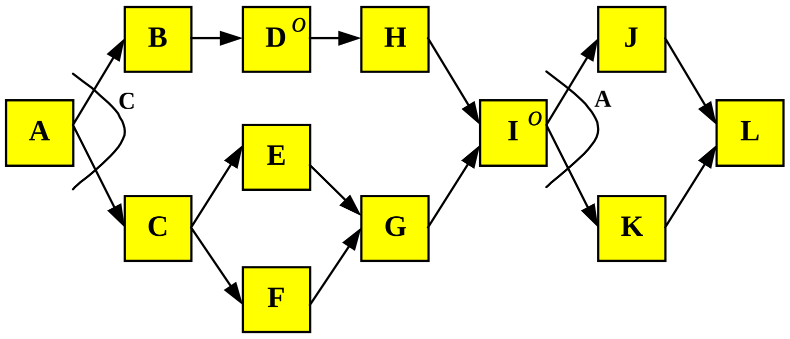

Figure 1: Example workflow graph

We should note that there exists an important difference between conditional and alternative execution of activities. In the conditional case, the activity that is executed next depends on data and state generated during the execution of the workflow process instance. In the alternative case, the activity that is executed next depends on policies and information that is shared by all instances of the same workflow process. This implies that any alternative will lead to a valid workflow execution and, for time management, when the schedule is tight, the alternative with the shortest execution time can always be chosen (e.g. FedEx Express is chosen for international shipment instead of the regular postal service).

Due to conditional and alternative structures, different sets of activities are executed, depending on the case data of the workflow and/or on the policies and choices of workflow participants. Each different set of activities that corresponds to a valid execution is referred to as *workflow instance type*. For example, *A*,*B*,*H*,*I*,*K*,*L* is a workflow instance type of the workflow shown in Figure 1.

2.2 Execution Durations and Deadlines

In order to represent time information, we need to augment the workflow model with the following basic temporal types*: time points*, *durations*, and *deadlines*. For the sake of simplicity, we assume that all time information is given in some basic time units.

Given a workflow schema, a workflow designer can assign execution durations and deadlines to individual activities and to the whole workflow process [13,15,18,23]. These durations can be either calculated from past executions, or they can be assigned by specialists based on their experience and expectations. In addition, multiple execution durations may be assigned to an activity. Typically, the most common duration values used are minimum, maximum, and average.

Activity and process deadlines, on the other hand, correspond to maximum allowable execution times for activities and processes, respectively. In the remainder of this paper, we refer to these deadlines as *explicit deadlines*. At process build-time, these deadlines are specified relative to the beginning of the process, using some time granularity, e.g. 2 hours, 5 minutes, or by Wednesday. At process instantiation-time, a calendar is used to convert all relative deadlines to absolute time points, modify the assigned deadlines, or assign new deadlines.

It is important to note that activity durations and deadlines may not be the same, which is how they are always treated by some of the existing workflow management systems. Distinguishing between the two is beneficial for cases where the actions taken when a deadline is missed have a high cost associated with them (e.g. rollback of the entire process). In such cases, when an activity takes longer to execute than the duration assigned to it in the workflow schema, preemptive steps can be taken to assess deadline satisfiability, modify workflow parameters, and alert appropriate agents and process managers.

Deadlines do not have to be associated with every activity of a workflow schema. However, it is extremely beneficial to assign deadlines to all activities. The most compelling reason for this is the ability to monitor the execution progress of activities and processes so that preemptive actions are taken when delays are developed. We present how these deadlines, referred to as *internal deadlines*, are computed at process build- and instantiation-time and used at run-time in the sequel.

2.3 Explicit Time Constraints

Many time constraints are derived implicitly from control dependencies and activity durations. They arise from the fact that an activity can only start when its predecessor activities have finished. Such constraints are called *structural time constraints* because they reflect the control structure of the workflow. On the other hand, workflow designers can specify *explicit time constraints* based on organizational rules and business policies, laws and regulations, service-level agreements, and so on. Examples of such constraints include: (1) an invitation for a meeting has to be mailed to the participants at least one week before the meeting; (2) after a hardware failure is reported, a service team should be at the customer's site within 4 hours; (3) vacant positions can be announced on the first

Wednesday of each month; (4) inventory checks should finish by December 31st; (5) loans above USD 1M are approved during scheduled meetings of the board of directors.

Such explicit constraints are either temporal relations between events or bindings of events to certain sets of calendar dates. In workflow systems, these events correspond to two main events that are associated with an activity: *start* and *end*. The start event denotes the start of the activity, while the end event denotes its completion. For temporal relationships between events, the following time constraints can be defined, assuming that δ corresponds to a relative time duration.

- **Lower-bound constraint**: The time distance between source event *s* and destination event *d* must be greater than or equal to δ. The notation used is $lbc(s,d,\delta)$.
- **Upper-bound constraint**: The time distance between source event *s* and destination event *d* must be smaller than or equal to δ. The notation used is $ubc(s,d,\delta)$.

An example of a lower-bound constraint includes a legal workflow with activities of serving a warning and closing a business, with the requirement that a certain time period passes between serving the warning and closing the business. Another example is a chemical process control workflow where a reaction is initiated only when certain time passes after the start of another reaction. Upper-bound constraints are even more common. The requirement that a final patent filing is done within a certain time period after the preliminary filing and time limits for responses to business letters provide typical examples of such constraints.

To express constraints that bind events to specific calendar dates, an abstraction that generalizes a typically infinite set of dates (i.e. “every other Monday“ or “every 5th workday of a month“) is required.

- **Fixed-date object**: A fixed-date object is an abstract data type *T* with the following methods: *T.next(D)* and *T.prev(D)* return, respectively, the next and previous valid dates after an arbitrary date *D*; *T.period* returns the maximum distance between valid dates; and *T.dist(T′)* returns the maximum distance between valid dates in the given object and in another fixed-date object *T′*, having as default value *T.period*.

An example of such a fixed date object would be *em* (every Monday). The operation *em.next(D)* would return the date of next Monday after D, *em.prev(D*) would return the last Monday, and *em.period* would return 6 days. For the fixed-date object *efm* (every first of a month), the distance *em.dist(efm)* would be 30 days, while *efm.dist(em)* would be 6 days.

Having fixed-date objects in our disposal, we can now define *fixed-date constraints* as follows.

- **Fixed-date constraint**: To express a time constraint that binds an event E to some fixed date(s), we write *fdc(E,T)*, where *T* is a fixed-date object.

Although several fixed-date constraints could be associated with an activity, for simplicity, we assume that only one such constraint is used in the remainder of the paper.

2 Workflow Time Modeling

Workflow graphs can be extended to include time-related data. Since workflow graph nodes, which correspond to activities, have attributes associated with them, such as the role that is responsible for the enactment of the activity, one could easily model time-related data by adding more attributes. Moreover, time constraints between activities could be shown in the graph by additional edges, such as *time edges* [5]. In the rest of the paper, we focus on time-related activity attributes, and we discuss how these attributes can be used during the lifecycle of a process. A workflow graph that includes time information is referred to as *timed activity graph* or *timed graph*. Each activity node in an timed graph is called *timed activity node*.

As we mentioned in the previous section, each activity *A* has start and end events associated with it. Depending on the execution duration(s) associated with *A*, one could “attach“ several pairs of these events to *A*. For simplicity, however, we use the average execution time as the expected execution duration of an activity. Here, the start event can be computed when the end event is known and, thus, we only need to consider end events when modeling time constraints. For the computation of activity end events we developed an extension of the Critical Path Method (CPM) [29], a project planning method that is frequently used for project management.

Activity Name	
duration	*optional*
$\mathbf{E_{BF}}$	$\mathbf{L_{WF}}$
$\mathbf{E_{BS}}$	$\mathbf{L_{BS}}$
$\mathbf{E_{WF}}$	$\mathbf{L_{WF}}$
$\mathbf{E_{WS}}$	$\mathbf{L_{WS}}$

Figure 2: Timed activity node

Due to the conditional/optional execution of activities, the following time information can be associated with the end event of an activity *A*: E_{BS}, E_{WS}, E_{BF}, E_{WF}, $L_{BS,}$, L_{WS}, L_{BF}, *and* L_{WF}, as shown in Figure 2. Here, *E* stands for the earliest point in time *A* may end, while *L* stands for the latest possible point in time *A* can finish to ensure minimal execution time for the entire process. Since conditional branches may require different execution times, we use *B* to denote the best-case and *W* to denote the worst-case. Finally, optional and alternative activity executions are captured by *F* and *S*; *F* corresponds to an execution where optional activities are not executed and the fastest alternative is always selected, while *S* corresponds to an execution where all optional activities are executed and any alternative can be selected.

For example, E^{A}_{WS} corresponds to the earliest point in time *A* may finish when it belongs to the path that has the worst conditional branches, all optional activities are executed, and the slowest alternative is chosen. On the other hand, L^{A}_{WS} corresponds to the latest possible point in time activity *A* has to finish in order to minimize the execution of the entire process, assuming that the worst conditional branches will be followed, all optional activities will be executed, and any alternative can be selected in the remaining of the process. Figure 3 shows the result of calculating the E- and L-values for the workflow shown in Figure 1. These computations are outlined in the next section.

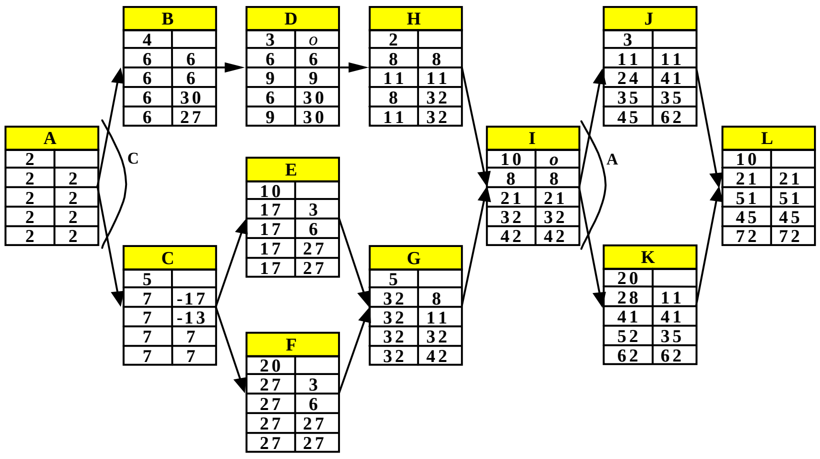

Figure 3: Example workflow graph including time information

The most important information present in this timed graph is activity E-values and, in particular, the E-values of activity *L*. This is because *L* is the last activity to be executed and, hence, its termination time point corresponds to the termination time point for the entire process. In particular, the earliest possible time for the entire workflow to end is 21, which corresponds to E^L_{BF}. This can happen when the conditional branch containing *B* is followed, activities *D* and *I* are not executed, and *J* is selected instead of *K*. On the other hand, if all optional activities are executed, the workflow can finish as early as 51 (E^L_{BS}) and as late as 72 (E^L_{WS}), depending on the specific alternative and conditional activity executions.

The L-values of an activity indicate whether there is a path containing this activity that may lead to a time error at process execution. In particular, if all L-values of an activity are greater than their corresponding E-values, there exist execution paths containing this activity that are likely to avoid time violations. However, if there exist L-values that are less than their matching E-values, then there exist paths that may lead to time violations. For example, activity *C* has negative L_{BF} and L_{BS} values in Figure 3. Therefore, if *C* is executed at run-time and the deadline of the entire process is set to 21, the deadline will be violated.

Finally, each node in a timed workflow graph includes aggregated time information across all workflow instance types. In the presence of conditionally executed activities with considerable variations in execution durations, this information might be too coarse grained for workflow designers. Here, the *unforlded timed workflow graph* may offer a solution. This graph contains exactly the same set of instance types as the original graph. However, it does not contain or-joins and has several termination nodes, one for each instance type. Once a workflow graph is unfolded, different time information can be assigned to activities in disparate instance types after their separating split node. We present more details about this in the following section.

3 Time Constraint Satisfiability

After activity durations and deadlines are assigned, time calculations are required for computing optimistic and pessimistic activity start and finish times, computing available slack time, updating existing deadlines, converting relative time information to absolute time points, and so on. Typically, the assignment of external deadlines is an iterative process. The designer first assigns activity durations and, then, she uses the time calculations at process build-time to compute the duration of the whole process and the relative position of all activities. The designer can then choose to set external deadlines to some of the activities and recompute the time information. If external deadlines cannot be met, the designer might modify the workflow structure, or change the deadlines.

4.1 Process Build-time

In this section, we outline a technique that can be used to verify time constraint satisfiability, i.e. it is possible to find a workflow execution that satisfies all constraints. A more detailed description of the technique can be found in [9,10].

4.1.1 Initial Computations

Initially, two passes over the workflow graph are performed, and the E- and L-values of all activities are computed using an extension of the CPM method. In particular, E-values of activities without predecessors are set to the durations of these activities, and a forward traversal of the workflow graph is done for computing the remaining E-values. Next, the L-values of activities without successors are set to their corresponding E-values, and a backward traversal of the workflow graph is done for computing the remaining L-values. During this traversal, if external deadlines exist, the L-values of the activities with such deadlines are set to these deadlines, which are assumed to be relative to the beginning of the workflow.

Once the above procedure is finished, the calculated E- and L-values reflect activity execution durations, lower-bound constraints, structural constraints, external deadlines, and explicit fixed-date constraints. An important aspect of the computation is the transformation of fixed-date constraints into lower-bound constraints using worst-case estimates. This mapping is necessary because, at build-time, calendar values for the workflow execution are not available and, thus, we can only use information about the duration between two valid time points for a fixed-date object. In particular, a fixed-date constraint *fdc(a,T)* is mapped to a lower-bound constraint *lbc(b,a,δ)* for every *b* that is a predecessor of *a*. The value of δ depends on whether *b* has a fixed-date constraint itself or not. If *b* does not have a fixed-date constraint, then δ is equal to *a.d+T.period*, where *a.d* is *a's* execution duration. If *fdc(b,T′)* exists, then δ is set to *T.dist(T′)*.

4.1.2 Incorporation of Upper-bound Constraints

Once the above process is completed, upper-bound constraints are incorporated into the computed E- and L-values. A necessary condition for the constraint *ubc(s,d,δ)* to be satisfiable is that the *distance* between the E- and L-values of *s* and their corresponding E- and L-values of *d* is less than δ. This distance is the sum of the execution durations of the activities on the longest path between *s* and *d*. Since this distance only depends on the E- and L-values of *s* and *d*, a violated upper-bound constraint could be satisfied by changing these values in a consistent way, i.e. by increasing the E-values of *s* and decreasing the L-values of *d*. The details on how such changes are performed, as well as the algorithmic properties of our technique can be found in [10]. We should note, though, that the satisfaction of individual upper-bound constraints may lead to a violation of already incorporated upper-bound constraints. Therefore, when an upper-bound constraint is incorporated into the E- and L-values of activities, all previously incorporated upper-bound constraints should be validated again.

At the end of the build-time calculations, there exist at least two (possibly not distinct) valid workflow executions. These executions are obtained when all activities complete at their E-values or their L-values. There may be other valid combinations of activity completion times within (*E*,*L*) ranges. We will say that a timed graph *satisfies* a constraint if the executions in which either all activities complete at their E- or all activities terminate at their L-values are valid with respect to this constraint. In addition, by examining the L-values of an activity, one can determine if there is a path containing this activity that may lead to time error during process execution. In particular, if all L-values of an activity are greater than their corresponding E-values, there exist execution paths containing this activity that are likely to avoid time violations. However, if some L-values are less than their corresponding E-values, then there exist paths that may lead to time violations.

4.1.3 Conditional Executions

When explicit time constraints involve conditionally executed activities, it may be beneficial to consider some/all of the conditional paths in isolation. By doing so, we may be able to avoid superfluous constraint violations and scheduling conflicts during process execution. In general, the following issues need to be addressed when we derive timed graphs that violate explicit time constraints.

- *Checking individual constraints for violation may not be sufficient*. As shown in [10], a set of time constraints may not be satisfiable, even when each individual constraint is satisfiable. Consequently, the incorporation procedure should consider all constraints together;
- *Checking workflow instance types for constraint violation in isolation is not sufficient*. Instance types only differ after or-splits and have the same initial activities; these common initial activities should have the same E- and L-values. If we cannot find such E- and L-values in all instance types to satisfy the constraints, then it may not be possible to schedule the execution of this workflow so that all time constraints are met;
- *Incorporating upper-bound constraints using best-case values may not be meaningful*. When an upper-bound constraint exists between a conditionally executed activity *C* and a successor activity *G*, which is always executed, checking this constraint for the best-case is not possible when the E- and L-values of *G* do not depend on the best-case E- and L-values of *C*;
- *Checking violation of upper-bound constraints using worst-case values may lead to unnecessary rejections when the workflow has conditional branches.* Similar to the above case, when the worst-case E- and L-values of *C* do not contribute to the worst-case values of

another activity, we may find a constraint violation when trying to incorporate several constraints.

To meet these restrictions we employ the technique of unfolding workflow graphs, which was sketched in Section 3. The unfolded graph is equivalent to the original graph since every execution valid in one workflow specification is also valid in the other. Another advantage of unfolding a workflow graph is that different deadlines, i.e. worst-case E- and L-values, may be assigned to activities belonging to different instance types. Consequently, different deadlines for the final activities of some instance types may be computed. As shown in [27], combining the different deadlines for disparate instance types together with the probabilities of executing such instance types is beneficial.

Intuitively, the unfolded timed workflow graph is derived from the original workflow graph by duplicating the graph at or-joins. Therefore, two instance types share the same nodes before the first or-split that distinguishes them and have different nodes for activities thereafter, even for activities shared after the or-join that merges these instances in the original graph. In practice, however, the unfolding procedure is more complicated since all unconditionally executed parallel branches have to be closed with and-joins (see [8] for details).

After the construction of the unfolded workflow graph, the temporal constraints are mapped into this graph. The mapping is done in such a way that if a constraint with source *s* and destination *d* exists in the original graph, then this constraint exists between all copies of *s* and *d* that belong to the same instance type in the unfolded graph. Once we are done mapping the constraints, we can compute the timed workflow graph based on the unfolded graph and then incorporate the temporal constraints using a variation of the algorithms presented in [10].

While the above procedure addresses the constraint incorporation problems, it suffers from the potential explosion of the number of "duplicate" nodes in the unfolded graph, since it considers each instance type separately. This is not desirable when discriminating between instance types is not necessary because either there are no interfering constraints in these instance types or we can check the satisfiability of such constraints without unfolding. This problem is addressed by partially unfolding the graph.

Partial unfolding takes place when an upper-bound constraint is violated during its incorporation in the original time graph. In particular, when an upper-bound constraint is violated, we determine whether its source and destination nodes are connected via conditionally executed activities or they belong to the same workflow instance type. Here we partially unfold the workflow graph and, finally, we attempt the constraint incorporation procedure again. If a constraint is violated and its source and destination nodes cannot be used for unfolding, then we check whether there is an overlapping constraint and perform the unfold for the source and destination nodes of this constraint. See [8] for details on the procedures for partial unfold and determination where to partially unfold.

4.2 Process Instantiation-time

At process instantiation-time, all relative E- and L-values should be replaced with absolute time points, and all time constraints should be chekked for satisfiability. Depending on the kind of the workflow, replacing relative E-and L-values with absolute time points may not be a straightforward process. The main factors that complicate this process are agent loads and availability and activity executions in many time zones.

Agent load and availability are hard to compute, especially when agents are humans. What complicates these computations is the fact that agents may participate in multiple workflow processes and have different availability schedules. One way to address this issue is by computing service time probability distributions for each agent. Such computations can be performed using the information that is usually logged by WFMSs about activity and process executions. When several agents can execute an activity, the minimum, maximum, or average service time can be used for computing the absolute E- and L-values of the activities executed by these agents.

With regards to activity executions in different time zones, multiple calendars will have to be used during the mapping to absolute time points. For example, consider an upper-bound constraint between *s* and *d* with δ being *5 business days*. Depending on the geographic location where *s* and *d* are executed, *5 business days* may correspond to many more than 5 calendar days.

Once the mapping to absolute time points is completed, an overall deadline for the entire process may be specified. Then, the same technique that was used during build-time calculations can be used here. Figure 4 shows the workflow graph after deadlines for activities *L*, *H*, and *G* are assigned. By looking at the values for activity *A*, we can conclude that it is possible to meet all deadlines. However, if at conditional branches longer paths are followed, then it is necessary to skip optional activities or select faster alternatives.

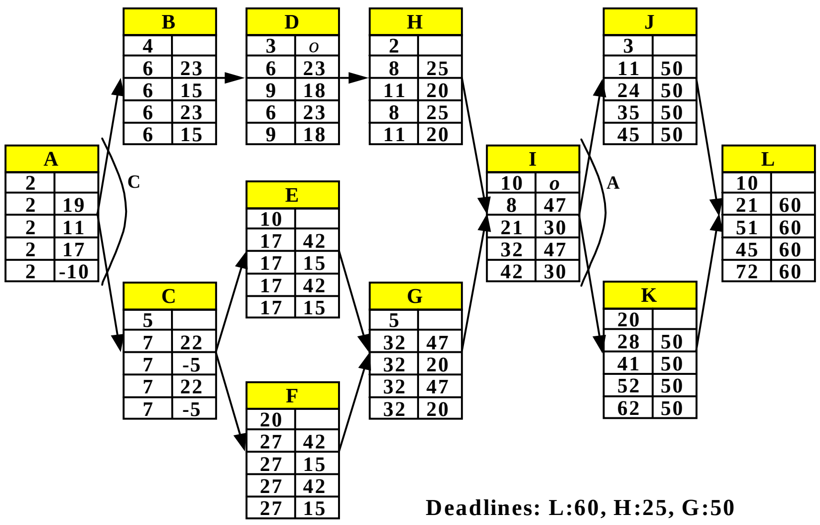

Figure 4: Example workflow graph with deadlines

If all L-values are negative, we cannot make it and we should raise a time exception. At this point, E-values are not really needed since the L-values are affected by the external deadlines. However, E-values could be used for performing agent load analysis. This can be done by checking the activities that are/will be assigned to an agent and the E-values for these activities. This topic, not discussed further in this paper, is subject of ongoing research to improve the forecast of delays in workflow executions.

4.3 Process Execution

At run-time, the workflow system should monitor the temporal status of a workflow so that an alert for a possible violation of a time constraint is raised early enough for pre-emptive steps to be taken. Furthermore, since the timed graph only guarantees that there exists a correct execution between the E- and L-values of each activity, recomputations of the timed graph are necessary to accomodate the time information, when activities are actually executed, and to use the information about decisions made at split points.

4.3.1 Process State Monitoring

During the execution of a workflow instance, actual activity execution times may vary considerably from the estimated execution times used in the time computations during process build and instantiation. When the execution time is less than the estimated execution, slack time becomes available. On the other hand, when the execution takes longer than the estimated execution, slack time for future activities may be reduced. In addition to the slack time generated when activities take less time to finish, slack time may be available due to the following.

- The deadline assigned to the workflow process is greater than the L-values of all activities that signal the end of the process, i.e. they have no successors;
- Activities belonging to parallel branches may have different execution characteristics. Since the longest branch determines the execution of all parallel branches belonging to the same unconditional split point, shorter branches have slack available to them;
- In conditional and alternative structures, slack is generated from the difference in the duration of different paths;
- When an optional activity is not executed, its estimated execution time becomes the available slack for its successor activities.

Given the current absolute time point, *now*, the estimated duration of an activity *A*, and the L-values of *A*, we can assess the state of the workflow instance containing *A* with respect to its execution progress as follows.

- If $now+duration(A) \leq L_{WS}^{A}$, the process is running smoothly and all deadlines will be met, given that remaining activities finish within their expected execution times;
- If $L_{WS}^{A} \leq now+duration(A) \leq L_{WF}^{A}$, the process can still meet all deadlines. However, it might be necessary to drop optional activities or choose faster alternatives;
- If $now+duration(A) \leq L_{BF}^{A}$, there is still a chance that the workflow finishes in time. However, this depends on the executed conditional branches;

- If $L^A_{BF} \leq now+duration(A)$, then it is possible to meet the deadlines only if the remaining activities finish faster than expected.

We should note that since externally defined deadlines are already taken into account during the construction of the timed graph, we do not have to consider these deadlines as long as L_{BF} can be met. If L_{BF} is missed, escalation is invoked. Based on the above observations, we can summarize the status of a workflow using the following states.

green: We expect to finish the workflow in time without dropping any of the optional activities or changing the alternative selection policies;

yellow: Although we may still be able to finish in time, we may have to eliminate some of the optional activities or select fast alternatives. In particular, before launching an optional activity, a decision has to be made whether the activity should be executed. Similarly, a decision needs to be made regarding the selection of the alternative activity to execute next. The rest of the activities are executed normally in this state;

red: The threat of missing a deadline is great and a time error should be raised to trigger escalation actions.

To monitor the state at which a process instance is currently operating, we can use two threshold values for each activity, L_{GY} and L_{YR}. L_{GY} signals the change from a green state to a yellow state. L_{YR} signals the change from a yellow state to a red state. Default values for these thresholds are set as follows: $L^A_{GY} = L^A_{WS}$ and $L^A_{YR} = L^A_{WF}$. These values are conservative choices, where no risk concerning alternative paths is taken. However, these threshold values should take into account the variance in activity durations, the proportion of best- and worst-cases, and the willingness to accept risks and, thus, are influenced by more information than is usually available in workflow systems. It is an important tuning knob for time management, and we believe that it should be the responsibility of a process manager to set these values and adjust them accordingly.

4.3.2 Time Computations

The above thresholds could be treated as internal activity deadlines (i.e. deadlines not assigned at build or instantiation-times). In particular, the deadlines of all non-optional activities could be set to L_{YR}, while the deadlines of optional activities could be set to L_{GY}. Note that for optional activities we use L_{GY} because a decision has to be made before launching such activities, according to the discussion presented in the description of the yellow state. However, it may be beneficial to assign different internal activity deadlines than the above threshold values when these deadlines can influence the sequence in which activities are selected from worklists and, hence, influence when activities are executed. For instance, in workflow systems where the shortest deadline first scheduling policy is used by the engine or the workflow participants can choose the next activity to execute from their worklists, strict internal deadlines can be used to accelerate process execution and create slack that can be used to address unexpected delays and exceptions in the future. If these deadlines cannot be met, deadline extension can be granted based on the current state of the process and the available slack.

Possible alternatives for computing these internal deadlines are the *no slack* and *proportional slack* policies described in [28]. In the former case, the internal deadline is set to the duration of the activity. In the later case, the duration is extended by a fraction of the available *slack* according to the *proportion* of the duration of the actual activity to the duration of the rest of the workflow. If the internal deadline does not influence the order in which activities are selected from worklists, which is the case when FIFO is used, the internal deadlines are not necessary and the threshold values introduced above can be used. This policy corresponds to the *total slack* policy presented in [28]. For these worklist selection strategies, the deadline is only necessary to determine when an escalation has to be raised.

With regards to time constraints and their satisfiability, we may have to delay the execution of some of the activities that are either sources of upper-bound constraints or destinations of lower-bound constraints. Even when we can immediately start the execution of an activity that is the source of some upper-bound constraint, it can be advantageous to delay its execution so that the remaining activities have more slack time, as shown in [10]. While existing work [26,27,28] can be used for distributing available slack times to activities, selecting an optimal delay value for an activity and allowing an activity to finish before its Evalue are part of on-going work.

When an activity *a* finishes in the interval (*a.E,a.L*), we may have to recompute the timed graph and reincorporate upper-bound constraints before modifying the L-values of the ready activities according to the slack distribution algorithm. In addition, the re-computation of the timed graph should use the L-values of any active activities for computing E-values in order to avoid upper-bound constraint violations. Finally, time constraint satisfiability tests would have to be performed when explicit time constraints involve activities that belong to loops. Here, during each loop iteration, time calculation would have to take place.

4 Handling Missed Deadlines

When a deadline is missed, a time failure is generated and escalation actions are taken. These escalation actions depend on the state of the workflow process (green, yellow, or red), and some of the possible alternatives are the following.

- **Deadline extension:** When an internal deadline is missed while the process is in either the green or the yellow state, the deadline may be extended. For non-optional activities, the upper-bound for the new internal deadline is L_{YR}. For optional activities, the upper-bound for the new internal deadline is L_{YR}, according to the discussion presented in the previous section. Extending internal deadlines is helpful when the proportional slack or the no slack strategies are followed during deadline assignment;
- **Alternative selection:** When the process is in the yellow state and its internal deadline is missed, besides extending its deadline, the selection policy for future alternative activities may be changed to favor alternatives with faster execution times. Of course, the above is beneficial only when the process deadline can be met with these changes. The preemptive escalation work of [27,28] can be used for determining this;
- **Option removal:** If no deadline extension can be granted and no alternative selection policy can be altered to preserve the process deadline, future optional activities can be eliminated. Actually, these optional activities are marked as dropped, and the decision to drop them is made when they are about to be scheduled for execution;

- **Time error:** If the process is in the red state, a timing exception has to be raised to escalate the problem. Here, recovery may be automatically invoked (similar to [11]) or human interaction may be required to proceed. In the latter case, there are several options available to process managers in order to regain a valid workflow state. The workflow schema can be dynamically changed (e.g. by parallelizing sequential activities), activity priorities can be raised to speed up execution, or deadlines can be renegotiated.

The escalation strategy tries to avoid higher escalations as long as possible. The threshold values between the timing states defined above are again used for determining the escalation level. Pro-active actions like avoiding alternative branches or skipping optional activities are delayed as long as possible. When such pro-active means are taken, the timed graph has to be recomputed to reflect the changed workflow.

6 Schedule-based Executions

The execution of a workflow instance requires re-computation of the timed graph after the completion of an activity that is the source of a lower-bound constraint or has a successor that is the source of an upper-bound constraint. These re-computations could be avoided by sacrificing some flexibility in the timed graph. Recall that the timed graph specifies ranges for activity completion times such that there *exists* a combination of activity completion times that satisfies all timing constraints and in which each completion time is within the range of its activity. Run-time re-computation was required because once completion time for finished activities has been observed, not all completion times within the ranges of the remaining activities continue to be valid.

We define a *schedule* to be a (more restrictive) timed graph in which *any* combination of activity completion times within [*E,L*] ranges satisfies all timing constraints. In other words, given a schedule, no violations of time constraints occur as long as each activity *a* finishes at time within the interval [*a.E,a.L*]. Consequently, as long as activities finish within their ranges, no timed graph re-computation is needed. Only when an activity finishes outside its range the schedule for the remaining activities must be recomputed.

It follows directly from the schedule definition that, for every upper-bound constraint *ubc(s,d,δ)*, $s.E+\delta \le d.L$ and for every lower-bound constraint *lbc(s,d,δ)*, $s.L+\delta \le d.E$; the reverse is also true, i.e. the timed graph that satisfies these properties is a schedule. From the way we compute E- and L-values for the activities in a timed workflow graph, the E- and L-values already qualify as schedules. Consequently, when every workflow activity finishes execution at its E-value, there is no need to check for time constraint violations. The same is true when activities finish execution at L-values. In the remainder of the section, we present two approaches for computing less strict schedules. Both algorithms start with the timed graph obtained at build- or instantiation-time. We should note that these techniques correspond to, somehow, extreme schedules and additional work is required for generalizing them.

- **Early scheduling:** This technique maintains the computed E-values and attempts to change the L-values so that $s.E+\delta \ge d.L$ and $s.L+\delta \le d.E$ hold for all *ubc(s,d,δ)* and *lbc(s,d,δ)*, respectively, without violating any constraints. For *ubc(s,d,δ)* with $s.E+\delta < d.L$, we change *d.L to be* $s.E+\delta$. For *lbc(s,d,δ)* with $s.L+\delta > d.E$, we change *s.L* to be $d.E-\delta$. Then, we recompute the timed activity graph and attempt to satisfy all constraints;
- **Late scheduling:** This technique maintained the computed L-values and attempts to change the E-values so that $s.E+\delta \ge d.L$ and $s.L+\delta \le d.E$ hold for all *ubc(s,d,δ)* and *lbc(s,d,δ)*, respectively, without violating any constraints. For *ubc(s,d,δ)* with $s.E+\delta < d.L$, we change *s.E to be* $d.L+\delta$. For *lbc(s,d,δ)* with $s.L+\delta > d.E$, we change *d.E to be* $s.L+\delta$. Then, we recompute the timed activity graph and attempt to satisfy all constraints.

The main advantage of schedule based executions is the reduction of necessary time calculations at run-time which makes this strategy in particular valuable for production workflows with large numbers of instances. An additional advantage is the better predicatbility of workloads which makes the planning of resources easier. In particular, we envision a situation where a possible future workflow engine can interact with calendars of workflow participants, checking for the availability of work-time and inserting possible (depending on conditionals) future tasks. The price for the reduction of recalculations of timed graphs is, however, a decrease of flexibility.

7 Related Work

Although the area of time management has received a lot of attention in areas such as project management, job-shop scheduling, and active databases, currently available commercial workflow products provide little support beyond simple monitoring of activity deadlines. On the other hand, workflow research recently stared addressing time management issues. In particular, an ontology of time for identifying time structures in workflow management systems is developed in [19]. The authors represent time aspects within a workflow environment by using the Event Condition Action (ECA) model found in active database management systems, and they discuss special scheduling aspects and basic time-failures, in particular, to reflect changes of the process. In contrast, our goal is to capture time information at build-time, monitor process execution at run-time, and react to time failures *without* modifying the business process model. In this, it is somewhat similar to scheduling in real-time systems [1,17,24]. However, real-time systems use deadlines for scheduling system components such as CPU and I/O. We view scheduling and internal deadline assignment and adjustment as complimentary mechanisms.

In [20,21], the authors studied the problem of how the deadline of a real-time activity is automatically translated to deadlines for all sequential and parallel subtasks constituting the activity. Each subtask deadline is assigned just before the subtask is submitted for execution, and the algorithms for deadline assignment assume that the *earliest deadline* first scheduling policy is used. While our work has similarities with the above work, there are several important differences. In particular, we treat alternative, conditional, and optional activities. Also, we offer techniques for building the timed graph at process build-time and using the graph for arriving at a process deadline. Finally, our work supports the assignment of external deadlines to individual activities as well as to the entire process.

In [26,27,28], the authors propose the use of static data (e.g. escalation costs), statistical data (e.g. average activity execution time and probability of executing a conditional activity), and run-time information (e.g. agent worklist length) to adjust activity deadlines and estimate the remaining execution time of workflow instances. However, this work can be used only at run-time and, furthermore, it does not address explicit time constraints.

In [2], the author proposes the integration of workflow systems with project management tools to provide the functionality necessary for time management. However, these project management tools do not allow the modeling of explicit time constraints and, therefore, they do not provide any means for their resolution.

In [30], the authors present an extension to the net-diagram technique PERT to compute internal activity deadlines in the presence of sequential, alternative, and concurrent executions of activities. Using this technique, business analysts provide estimates of the best, worst, and median execution times for activities, and the β-distribution is used to compute activity execution times as well as shortest and longest process execution times. Having done that, time constraints are checked at build-time and escalations are monitored at run-time. Our work extends this work by handling both structural and explicit time constraints at process build and instantiation-times, and enforcing these constraints at run-time.

In [12,31], the notion of explicit time constraints is introduced. Nevertheless, this work focused more on the formulation of time constraints, the enforcement of time constraints at run-time and the escalation of time failures within workflow transactions [7]. Our work follows the work described in [12,31] and extends it with the incorporation of explicit time constraints into workflow schedules.

This paper extends the time modeling and management technique presented in [10]. In particular, we extend the expressiveness of the workflow model by augmenting it with procedures for dealing with conditional executions. Consequently, we had to extend the computation of timed graphs and the incorporation algorithm for explicit time constraints to the increased expressiveness and complexity of conditional constructs.

In [5], the authors describe some of the time-related functionality of the $ADEPT_{time}$ workflow management system. As part of the time functionality, minimal and maximal durations may be specified for each workflow activity. In addition, time dependencies between workflow activities may be defined. These dependencies are the same as the lower- and upper-bound constraints we presented in this paper, and they are modeled using an additional edge that links the activities involved in such constraints. At build-time, the existence of a valid time schedule is chekked (i.e. an assignment of absolute start and finish times so that all constraints are satisfied). Start and finish times for activities are calculated at run-time using the Floyd-Warshall algorithm, and users are notified when deadlines are going to be missed.

Finally, the work presented in [25] is close to our work. However, there are important differences between the two. In contrast to [25], we do not consider time constraints in isolation and provide solutions for overlapping, interleaving, and interfering constraints. As we demonstrated in [10], a set of time constraints may be unsolvable (i.e. there is no instance of a workflow that does not violate at least one time constraint) even when every single constraint is solvable in isolation. In addition, our techniques are pro-active in nature, and they attempt to modify the E- and L-values of activities to make constraints satisfiable.

8 Conclusions

Even though currently available workflow management systems (WFMSs) offer sophisticated modeling tools for specifying and analyzing workflow processes, their time management support is still rudimentary. Existing time management functionality mainly addresses process simulations (to identify process bottlenecks, analyze execution durations, etc.), assignment of deadlines to activities, and triggering of process-specific exception-handling activities (referred to as escalations) when deadlines are missed during process execution. In this paper, we addressed the crucial role of time management in the life-cycle of workflow processes. In particular, we described how structural (i.e. execution order dependent) and explicit (i.e. fixed-date, periodic, upper- and lower-bound) time constraints can be modeled during process definition, validated during modeling and instantiation-times and, finally, monitored and managed during execution time.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: a performance evaluation. In *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 1-12, Los Angeles, CA, 1988.

[2] C. Bussler. Workflow Instance Scheduling with Project Management Tools. In 9th *Workshop on Database and Expert Systems Applications DEXA'98*, Vienna, Austria, IEEE Computer Society Press, 1998.

[3] CSE Systems. *Benutzerhandbuch V 4.1 Workflow*. CSE Systems, Computer & Software Engineering GmbH, Klagenfurt, Austria, 1996.

[4] CSE Systems Homepage. http://www.csesys.co.at/, February 1998.

[5] P. Dadam, M. Reichert, and K. Kuhn. Clinical workflows the killer application for processoriented information systems. In 4th International Conference on Business Information System (BIS 2000), pages 36--59, Poznan, Poland, 2000.

[6] J. Eder, H. Groiss, and W. Liebhart. The workflow management system Panta Rhei. *In NATO Advanced Study Institue on Workflow Management Systems and Interoperability*, Istanbul, Turkey, August 1997.

[7] J. Eder and W. Liebhart. Workflow Transactions. In P. Lawrence, Editor, *Workflow Handbook 1997*. John Wiley, 1997.

[8] J. Eder , W. Gruber and E. Panagos. Temporal modeling of workflows with conditional execution paths. In *11th In ternational Conference on Database and Expert Systems Applications DEXA 2000*, London, Greenwich, 2000.

[9] J. Eder, E. Panagos, H. Pozewaunig, and M. Rabinovich. Time Management in workflow system. In W. Abramowicz and M.E. Orlowska, editors, *BIS'99 3rd International Conference on Business Information System*, p. 265-280. Springer Verlag London Berlin Heidelbeg, 1999.

[10] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. *In Proc. International Conference CAiSE'99*. Springer Verlag, LNCS, 1999.

[11] J. Eder and W. Liebhart. Workflow recovery. *In First IFCIS* International Conference on Cooperative Information Systems *(CoopIS 96)*, Brussels, Belgium, Jun 1996. IEEE Computer Society Press.

[12] J. Eder, H. Pozewaunig, and W. Liebhart. Timing issues in workflow management systems. Technical report, Institut f'ür Informatik-Systeme, Universität Klagenfurt, 1997.

[13] TeamWare Flow. Collaborative workflow system for the way people work. P.O. Box 780, FIN00101, Helsinki, Finland.

[14] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119-153, 1995.

[15] HewlettPackard Company, 3000 Hanover Street, Palo Alto, CA 94304, USA. *HP Changengine Process Design Guide*, 3.3 edition, 2000. http://www.ice.hp.com/cyc/af/00/1010274.dir/cpdg.pdf.

[16] D. Hollingsworth. The workflow reference model. Draft 1.1 TC001003, Workflow Management Coalition, July 1995.

[17] J. Huang, J.A. Stankovic, D. Towsley, and K. Ramamritham. Experimental evaluation of realtime transaction processing. In Procee-ding of the 10th RealTime Systems Symposium, December 1989.

[18] InConcert. Technical product overview. XSoft, a division of xerox. 3400 Hillview Avenue, Palo Alto, CA 94304. http://www.xsoft.com.

[19] Heinrich Jasper and Olaf Zukunft. Zeitaspekte bei der Modellierung und Ausf¨uhrung von Workflows. In S. Jablonski, H. Groiss, R. Kaschek, and W. Liebhart, editors, *Geschäftsprozeßmodellierung und Workflowsysteme*, Volume 2 of *Proceedings Reihe der Informatik '96*, pages 109 -- 119, 1996.

[20] B. Kao and H. GarciaMolina. Deadline assignment in a distributed soft realtime system. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 428--437, 1993.

[21] B. Kao and H. GarciaMolina. Subtask deadline assignment for complex distributed soft realtime tasks. Technical Report 931491, Stanford University, 1993.

[22] P. Lawrence. *Workflow Handbook* 1997. John Wiley & Sons, 1997.

[23] F. Leymann and D. Roller. Business process management with flowmark. In *Proceedings of the 39th IEEE Computer Society International Conference*, pages 230--233, San Francisco, California, February 1994. http://www.software.ibm.com/workgroup.

[24] C.L. Lin and J. Layland. Scheduling algorithms for multiprogramming in hard realtime environments. *Journal of the Association of Computing Machinery*, 20(1):46--61, January 1973.

[25] O. Marjanovic and M. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2), May 1999.

[26] E. Panagos and M. Rabinovich. Escalations in workflow management systems. In *DART Workshop*, Rockville, Maryland, November 1996.

[27] E. Panagos and M. Rabinovich. Predictive workflow management. In *Proceedings of the 3rd International Workshop on Next Generation Information Technologies and Systems*, Neve Ilan, ISRAEL, June 1997.

[28] E. Panagos and M. Rabinovich. Reducing escalationrelated costs in WFMSs. In *NATO Advanced Study Institue on Workflow Management Systems and Interoperability*, Istanbul, Turkey, August 1997.

[29] Susy Philipose. *Operations Research A Practical Approach*. Tata cGrawHill, New Delhi, New York, 1986.

[30] H. Pozewaunig, J. Eder, and W. Liebhart. ePERT: Extending PERT for workflow management systems. In *First European Symposium in Advances in Databases and Information Systems (ADBIS*), St. Petersburg, Russia, 1997.

[31] Heinz Pozewaunig. Behandlung von Zeit in Workflow--Managementsystemen Modellierung und Integration. Master's thesis, University of Klagenfurt, 1996.

[32] SAP Walldorf, Germany. *SAP Business Workflow@OnlineHelp*, 1997. Part of the SAP System.

[33] Ultimus. Workflow suite. Business workflow automation. 4915 Waters Edge Dr., Suite 135, Raleigh, NC 27606. http://www.ultimus1.com.

[34] Workflow Management Coalition, Brussels, Belgium. *Glossary: A Workflow Management Coalition Specification, November 1994.*