Temporal Modeling of Workflows with Conditional Execution Paths

Johann Eder[1], Wolfgang Gruber[1], and Euthimios Panagos[2]

[1] Department of Informatics-Systems, Univ. Klagenfurt, A-9020 Klagenfurt, Austria
{eder,gruber}@isys.uni-klu.ac.at
[2] AT&T Labs - Research, 180 Park Avenue, Florham Park, NJ 07932
thimios@research.att.com

Abstract. In this paper, we present a novel technique for modeling, checking, and enforcing temporal constraints in workflow processes containing conditionally executed activities. Existing workflow time modeling proposals either do not discriminate between time constraints that apply to disparate execution paths, or they treat every execution path independently. Consequently, superfluous time constraint violations may be detected at modeling time, even when each execution path does not violate any constraints. In addition, scheduling conflicts during process execution may not be detected for activities that are common to multiple execution paths. Our approach addresses these problems by (partially) unfolding the workflow graph associated with a process that contains conditionally executed activities and, then, incorporating the temporal constraints in the time calculations performed on the unfolded graph.

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. 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 [GHS95,Law97,Wor94,Hol95]. However, existing WFMSs [LR94,InC,Flo] offer limited support for modeling and managing time constraints associated with processes and their activities [PEL97]. This support appears mainly in the form of monitoring activity deadlines [Sch96]. However, the consistency of these deadlines and the side effects of missing some of them are not addressed.

In process centered organizations, time management is essential for process modeling and management. Many business processes have restrictions such as limited duration of subprocesses, terms of delivery, dates of re-submission, or activity deadlines. Typically, time violations increase the cost of a business process because they lead to some form of exception handling [PR97b]. Therefore, a WFMS should provide the necessary information about a process, its time restrictions, and its actual time requirements to a process manager. In addition, the process manager needs tools to anticipate time problems, proactively avoid time constraints violations, and make decisions about the relative priorities of processes and timing constraints.

The notion of *timed workflow graphs* was introduced in [EPPR99], and it was shown how the time information represented in these graphs can be used at modeling, process instantiation, and execution times to manage workflow executions without time errors. In [EPR99], we introduced *explicit temporal constraints* and presented a technique for incorporating these constraints into timed workflow graphs. However, the technique presented in [EPR99] treated temporal constraints associated with parallel and conditionally executed activities in the same way. In this paper, we show that differentiating between these cases is crucial in avoiding many superfluous constraint violations, and we present the (partial) unfolding technique for handling them.

2 Workflows, Constraints, and Timed Graphs

In this section, we present the necessary definitions, assumptions, and methods used in the remainder of the paper.

2.1 Workflows

A workflow is a collection of *activities, agents,* and *dependencies* between activities. Activities correspond to individual steps in a business process, agents (software systems or humans) are responsible for the enactment of activities, and dependencies determine the execution sequence of activities and the data flow between them. We assume that workflows are *well structured.* A well-structured workflow consists of m sequential activities, $T_1 \dots T_m$. Each activity T_i is either primitive, i.e., it cannot be decomposed any further, or composite. A composite activity consists of n_i parallel conditional or unconditional sub-activities ${T_i}^1, \dots, {T_i}^{n_i}$, each of which is either primitive or composite. Typically, well structured workflows are generated by workflow languages that provide the usual control structures and adhere to a structured programming style of workflow definitions, such as Panta Rhei [EGL97].

Workflows are represented by *workflow graphs*, where nodes represent activities and edges correspond to dependencies between activities. An *and-split* node refers to an activity having several immediate successors, all of which are executed in parallel. An *and-join* node refers to an activity that is executed after all of its immediate predecessors finish execution. An *or-split* node refers to an activity whose immediate successor is determined by evaluating some boolean expression. An *or-join* node refers to an activity that joins all the branches after an or-split. Finally, similar to [MO99], a *workflow instance type* refers to workflow instances that contain exactly the same activities, i.e., for each or-split node in the workflow graph, the same successor node is chosen.

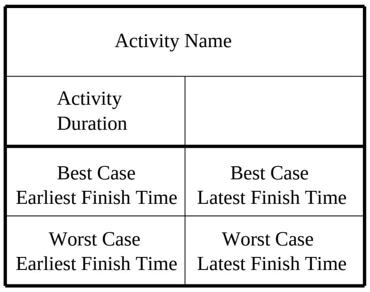

Fig. 1. Activity node of a timed workflow graph

2.2 Temporal Constraints

Time is expressed in some basic time units relative to a time origin, which is usually the start of the workflow. In addition, each activity has a duration which, for simplicity, is assumed to be deterministic. Activity durations and control dependencies between activities determine the *structural time constraints*. These constraints arise from the fact that an activity can only start when its predecessor activities are finished. In addition, workflow designers may specify *explicit time constraints*, i.e., temporal relations between start and end of (different) activities. These constraints are derived from organizational rules, laws, commitments, and so on (e.g., an appeal can be filed within 7 days after the verdict, a meeting invitation has to be sent to all participants at least one week before the meeting). In particular, the following explicit time constraints can be specified.

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

2.3 Timed Workflow Graphs

Our time constraint management techniques are based on the notion of a *timed workflow graph*, which extends the workflow graph by augmenting each activity node n with the following[1]. (Figure 1 shows the representation of such a node.)

- $n.E^{bc}$: The earliest point in time n can finish when the shortest path is chosen to reach n;
- $n.E^{wc}$: The earliest point in time n can finish when the longest path is chosen to reach n;
- $n.L^{bc}$: The latest point in time n has to finish in order to meet the overall deadline via the shortest path;

[1] Since activity durations are assumed to be deterministic, start times for activities are computed by subtracting their durations from their termination times.

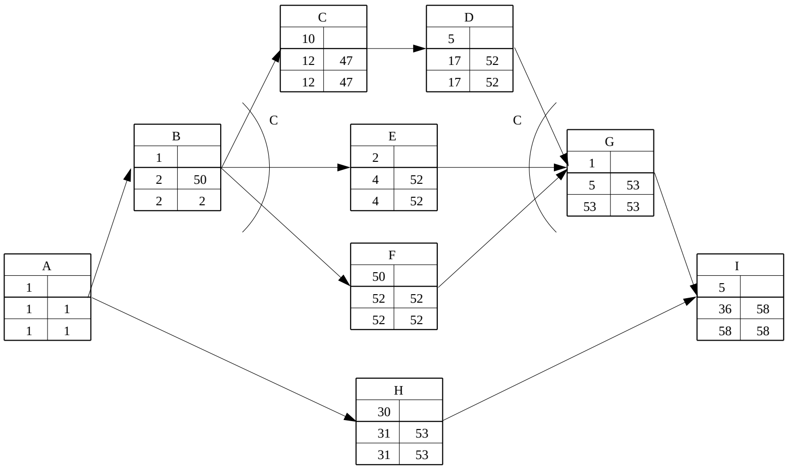

Fig. 2. Example timed workflow graph

- $n.L^{wc}$: The latest point in time n has to finish in order to meet the overall deadline via the longest path.

Without explicit time constraints, the above values can be computed by extending the Critical Path Method (CPM) [Phi86] to handle conditional execution paths [PEL97]. Table 1 shows the actual computations, where $d(n)$ denotes the execution duration of activity n. E-values are computed in a forward pass, with the E-values of the starting workflow activity being set to its duration. L-values are computed in a backward pass, with the L-values of the last workflow activity equal to its E-values.

Figure 2 shows the timed workflow graph we use in the rest of the paper. In this graph, activity A is followed by an and-split, having I as the and-join, and activity B is followed by an or-split, having G as the or-join. The values of node I show the duration of the workflow, i.e., $I.E^{bc}$ and $I.E^{wc}$ indicate that

FORWARD	**best case (bc)**	**worst case (wc)**
sequence	$E_j = E_\tau + d(j)$	$E_j = E_\tau + d(j)$
and-join	$E_j = \max(\{E_\tau + d(j)\})$	$E_j = \max(\{E_\tau + d(j)\})$
or-join	$E_j = \min(\{E_\tau + d(j)\})$	$E_j = \max(\{E_\tau + d(j)\})$
	$\forall$ immediate predecessor τ of j	
REVERSE	**best case (bc)**	**worst case (wc)**
sequence	$L_j = L_\tau - d(\tau)$	$L_j = L_\tau - d(\tau)$
and-split	$L_j = \min(\{L_\tau - d(\tau)\})$	$L_j = \min(\{L_\tau - d(\tau)\})$
or-split	$L_j = \max(\{L_\tau - d(\tau)\})$	$L_j = \min(\{L_\tau - d(\tau)\})$
	$\forall$ immediate successor τ of j	

Table 1. Calculation instructions for timed workflow graphs

the workflow execution may take between 36 and 58 time units. $G.E^{bc}$ indicates that G may finish after 5 time units (when E follows B), while $G.E^{wc}$ indicates that no path from A to G should take more than 53 time units. $B.L^{bc}$, tells us that if B is finished at time point 50, the workflow may still be able to terminate in time. From $B.L^{wc}$ we learn that if B is finished at time point 2, we can meet the overall deadline, irrespective of the conditionals.

3 Incorporating Explicit Time Constraints

Once the timed workflow graph is constructed, we can incorporate explicit time constraints into it by using the algorithms shown in [EPR99]. Lower bound constraints are incorporated during the construction of the timed workflow graph; they may increase E-values during the forward pass and decrease L-values during the reverse pass. On the other hand, the incorporation of upper-bound constraints should check for constraint violations; for $ubc(s, d, \delta)$, $s.E + \delta < d.E$ and $s.L + \delta < d.L$. When a constraint is violated, the E- and L-values of s and d are shifted in an attempt to satisfy the constraint, with the invariant that an E-value is not greater than its corresponding L-value.

During the incorporation of explicit constraints the semantics of the E-values change: the E-values mandate that activity terminations should not occur earlier than them in order to meet the time constraints. Therefore, the E- and L-values define the time interval during which an activity has to terminate. This time interval is referred to as the *life-line* of the activity.

However, [EPR99] does not discriminate between conditional and unconditional branches in the computation of worst case E- and L-values. While this ensures that the execution of the workflow will avoid violating temporal constraints when the incorporation algorithm succeeds, it is overly pessimistic. In particular, there are cases where execution without constraint violation is possible and the incorporation algorithm does not succeed due to interference of constraints on mutually exclusive conditional branches, as we show below.

In general, the following issues need to be addressed when we derive timed graphs that violate explicit time constraints.

1. *Checking individual constraints for violation may not be sufficient.* As shown in [EPR99], 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.
2. *Checking workflow instance types for constraint violation in isolation is not sufficient.* If two instance types only differ after or-splits, their 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. For example, consider two instance types for the workflow in Figure 2, where one includes C and D and results in $B.E^{wc} = 20$ and $B.L^{wc} = 40$, and the other one includes E and results in $B.E^{wc} = 5$ and $B.L^{wc} = 15$. Here, it is not possible to satisfy the time constraints for

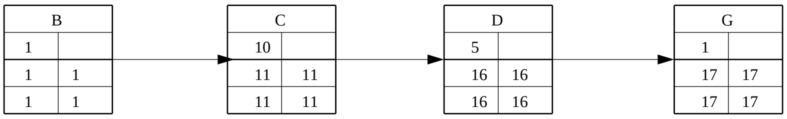

Fig. 3. Path with $ubc(B, D, 20)$ and $ubc(C, G, 15)$

either instance since B is executed in both, and the scheduling information for B, i.e., valid interval for B to terminate, is contradictory.

3. *Incorporating upper-bound constraints using best-case values may not be meaningful.* This can be seen by looking at the best-case values of C and G in Figure 2. The values of C do not really contribute to the values of G because they are dominated by those of E and F. Therefore, checking an upper-bound constraint between C and G for the best-case is not possible.
4. *Checking violation of upper-bound constraints using worst-case values may lead to unnecessary rejections when the workflow has conditional branches.* This can be seen by examining the case where $ubc(B, D, 20)$ and $ubc(C, G, 15)$ need to be incorporated into the timed graph shown in Figure 2. If we incorporate $ubc(B, D, 20)$ first, then $D.L^{wc}$ becomes 22 and, consequently, $ubc(C, G, 15)$ cannot be satisfied. However, each of these constraints is individually satisfiable, as shown in Figure 3, since the path containing C and D does not influence the computation of the worst case E- and L-values of nodes B and G.

4 Unfolded Workflow Graph

To address the time constraint incorporation problems, we construct the *unfolded timed workflow graph.* 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, we are able to assign different time information to activities in disparate instance types after their separating split node, and share the same time information for activities before this node. In this way, explicit temporal constraints that involve activities in different instance types no longer interfere and, thus, we may be able to satisfy all of them.

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 [PR97a], combining the different deadlines for disparate instance types 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 very first or-split that distinguishes them

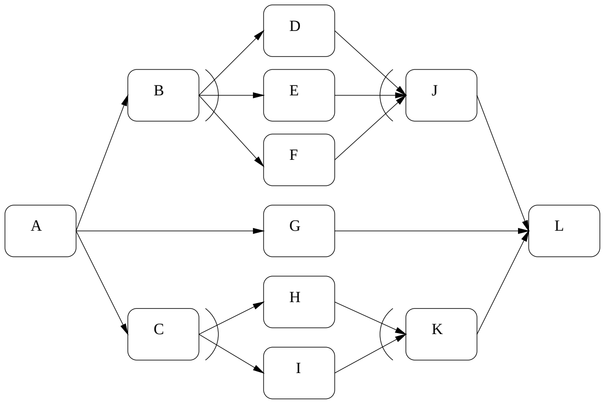

Fig. 4. Aggregated workflow graph

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 should be closed with and-joins. We explain how this is done below.

4.1 Unfolding Procedure

The procedure for generating an equivalent unfolded workflow graph U for a workflow graph G is as follows. First, the start node of G is copied into U. Then all nodes of G are visited in topological order. Copies of each node n of G, which is not an and-join, are inserted into U as many times as there are copies of the predecessors of n and the nodes are connected accordingly, such that each copy of node n is connected with exactly one copy of a predecessor of n and vice versa[2].

If n is an and-join node, then we place a copy of n in U for all valid predecessor combinations. These combinations are computed by constructing all combinations of copies of predecessor nodes of n in G, such that in each combination there is exactly one copy of each of these predecessor nodes (i.e., the Cartesian product of the copies of the respective nodes). Figure 5 shows the unfolded graph for the workflow graph shown in Figure 4.

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

[2] Copies of G's or-join nodes have exactly one predecessor node in U and, thus, there are not or-joins anymore.

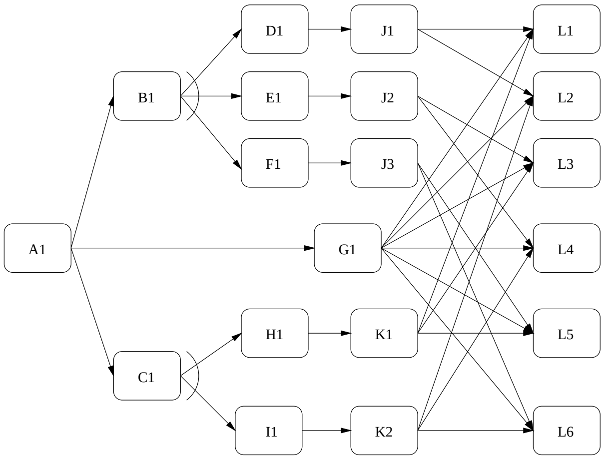

Fig. 5. Unfolded workflow graph

graph and then incorporate the temporal constraints using a variation of the algorithms presented in [EPR99].

While the above procedure addresses the constraint incorporation problems of Section 3, 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. To address this problem, we developed the *partial unfolding* technique.

4.2 Partial Unfolding of Workflow Graphs

Since constraints need not appear in every alternative path, we can unfold the workflow graph only where it is necessary to check constraints. We call such partially unfolded graphs "hybrid graphs". The procedure for partially unfolding a workflow graph G to a hybrid graph H begins by selecting a *hot-node*, with the side effect that all instance types going through the hot-node are factored out, or intuitively, the workflow graph reachable from the hot-node is duplicated. In principle, every node can be hot-node. For practical reasons, we require that a hot-node is an immediate predecessor of an or-join. In the next section we will show how hot-nodes are chosen, when a time constraint cannot be incorporated. Once a hot-node is identified, partial unfolding takes place as follows:

1. Copy the original graph G into H;
2. Insert an additional copy of all nodes reachable from the hot-node, together with a copy of all edges between such nodes;
3. Remove the edge from the copy of the hot-node to it's original successor and include an edge to the created copy of the successor node;

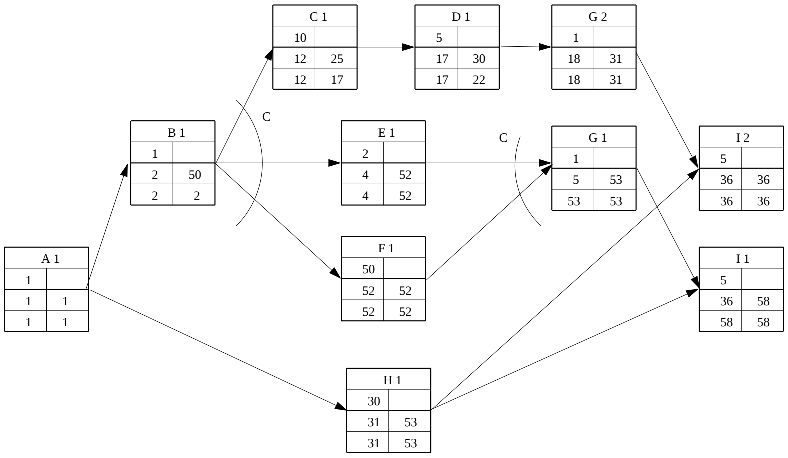

Fig. 6. Partially unfolded graph with $ubc(B1, D1, 20)$ and $ubc(C1, G2, 15)$

4. For all edges from a node n in G, which is not successor of the hot-node, to an and-join node a in the succession of the hot-node, include a copy in H;

Based on this procedure, we only have to partially unfold the workflow graph shown in Figure 2 at node D in order to check the satisfiability of $ubc(B1, D1, 20)$ and $ubc(C1, G2, 15)$. Figure 6 shows the resulting partially unfolded graph.

4.3 Computation of the Timed Graph

We first compute the timed graph using structural constraints and any explicit lower-bound constraints. Then, we attempt to incorporate all upper-bound constraints. 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 first determine the hot-nodes, then we partially unfold the workflow graph and, finally, we attempt the constraint incorporation procedure again.

For checking the satisfiability of time constraints, we extended the algorithm in [EPR99] to the full nodes (best/worst case). If a constraint $ubc(s, d, \delta)$ cannot be incorporated, the algorithm terminates with an error. In this case, we determine a node s' as the first successor node of s that immediately precedes an or-join and all paths from s to s' have the same number of or-joins and or-splits[3]. In the same way, we determine node d' for destination node d. If such nodes exist, we restart the incorporation procedure on the partially unfolded workflow graph that used these nodes as hot-nodes. Otherwise, either the path has been unfolded already, or the nodes do not have multiple copies in the unfolded graph.

[3] Intuitively, we search for the or-join reached from s that joins the conditional branch through s with its parallel branches and take the immediate predecessor of this or-join as s'.

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. An example for this procedure is given in the workflow shown in Figure 2 and the constraints $ubc(B, D, 20)$ and $ubc(C, G, 15)$. If $ubc(B, D, 20)$ is incorporated first, then $ubc(C, G, 15)$ cannot be incorporated since $D.L^{wc}$ is 22. Now we take D as hot-node and partially unfold the workflow graph, computing the timed hybrid graph, and incorporating the constraints there. The result of this procedure is the graph shown in Figure 6.

The algorithms for incorporating explicit time constraints in timed workflow graphs with (partial) unfolding have been implemented in a prototype for an extended workflow design tool. The prototype accepts workflow descriptions in the workflow definition language of the workflow system Panta Rhei [EGL97], extended with explicit time constraints. The algorithms for unfolding and partially unfolding workflow graphs as well as the algorithms for computing timed workflow graphs and incorporating explicit time constraints into these timed graphs are defined on these process definitions.

5 Related Work

Incorporating explicit time constraints into the modeling and management infrastructure of WFMSs has received very little attention from both workflow vendors and researchers. Among the work that is available in the literature, our work is closely related to [EPR99,MO99]. In particular, we extended [EPR99] to explicitly handle temporal constraints associated with conditionally executed activities by unfolding the timed workflow graph of a process. By doing so, we are able to avoid many superfluous constraint violations.

In contrast to [MO99], we do not consider time constraints in isolation and provide solutions for overlapping, interleaving, and interfering constraints. As we demonstrated in [EPR99], a set of time constraints may be unsolvable (i.e., there is no workflow instance 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 in order to make constraints satisfiable.

6 Conclusions

In this paper, we presented a new technique for modeling, checking, and enforcing temporal constraints in workflow processes containing conditionally executed activities. Our technique discriminates between time constraints that apply to disparate execution paths and, thus, it avoids the superfluous time constraint violations detected by existing techniques that treat these paths similar to those of unconditionally executed activities. In addition, our graph unfolding procedure and the incorporation of explicit time constraints into the unfolded graph avoid the problem of detecting scheduling conflicts when workflow instances are treated independently of each other.

References

[EGL97] Johann Eder, Herbert Groiss, and Walter Liebhart. The Workflow Management System Panta Rhei. In A. Dogac et al. (eds.), *Advances in Workflow Management Systems and Interoperability*. Springer Verlag, 1997.

[EPPR99] Johann Eder, Euthimios Panagos, Heinz Pozewaunig, and Misha Rabinovich. Time management in workflow systems. In *Business Information Systems BIS'99*, pages 265–280. Springer Verlag, 1999.

[EPR99] Johann Eder, Euthimios Panagos, and Misha Rabinovich. Time constraints in workflow systems. In *Proc. Int. Conf. CAiSE'99*. Springer Verlag, 1999.

[Flo] TeamWare Flow. Collaborative workflow system for the way people work. P.O. Box 780, FIN-00101, Helsinki, Finland.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation. In A. Elmagarmid, ed., *Distributed and Parallel Databases*, vol 3. Kluwer, 1995.

[Hol95] D. Hollingsworth. The workflow reference model. Draft 1.1 TC00-1003, Workflow Management Coalition, July 1995.

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

[Law97] P. Lawrence. *Workflow Handbook 1997*. John Wiley & Sons, 1997.

[LR94] 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.

[MO99] O. Marjanovic and M. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2), May 1999.

[PEL97] H. Pozewaunig, J. Eder, and W. Liebhart. ePERT: Extending PERT for Workflow Management Systems. In *First EastEuropean Symposium on Advances in Database and Information Systems ADBIS ́97*, 1997.

[Phi86] Susy Philipose. *Operations Research - A Practical Approach*. Tata McGraw-Hill, New Delhi, New York, 1986.

[PR97a] 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.

[PR97b] E. Panagos and M. Rabinovich. Reducing escalation-related costs in WFMSs. In A. Dogac et al.(eds.), *NATO Advanced Study Institue on Workflow Management Systems and Interoperability*. Springer Verlag, 1997.

[Sch96] G. Schmidt. Scheduling models for workflow management. In B. Scholz-Reiter and E. Stickel, editors, *Business Process Modelling*. Springer, 1996.

[Wor94] Workflow Management Coalition, Brussels, Belgium. *Glossary: A Workflow Management Coalition Specification*, November 1994.