Calculation of Delay Times for Workflows with Fixed-Date Constraints

Martin Bierbaumer, Johann Eder, Horst Pichler
Institute for Informatics-Systems, University of Klagenfurt, Austria

Abstract

Workflow participants are faced with the problem to select one out many work items in their worklist. The decision to execute one specific work item implicitly postpones the execution of all other items. This may lead to disproportionately increased execution durations of the associated workflows, especially if fixed-date constraints are defined on succeeding tasks. In this paper we show how to utilize a workflows control structure augmented with information about timing behavior to calculate what delay to expect when the execution of a work item is postponed to certain dates. This information can be used to assist the participant in order to increase the workflow throughput and avoid time-related escalations.

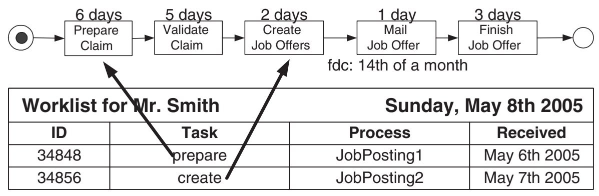
1 Introduction

Workflow systems execute tasks according to a process definition. Each manual task refers to a work item which is assigned to a workflow participants worklist. As a worklist usually holds multiple work items from different workflows the participant has to decide which work item to handle next. The decision of a workflow-participant to work on a particular work item implicitly holds the decision to postpone every other work item in the work list. This may have grave effects on the execution duration of the workflows to which these postponed work items belong. Common policies for this decision problem, like first-in first-out (FIFO) or earliest due date first, are known to be suboptimal [1]. Two reasons (among others) are that this policies do not consider the following: a) the postponement of a task may not immediately delay a workflow due to eventually existing buffer times, and b) even a slight postponement may lead to a disproportionately high delay due to fixed-date constraints [8] on succeeding tasks (e.g. 'meeting on 5th of a month').

Consider the workflow *JobPosting* to announce open positions of a big company in the local newspaper, as visualized in Fig. 1: A department initializes the process by generating a claim. At first the claim will be forwarded to the personnel division where it is *prepared* for further processing. Then the claim is *validated* by the personal manager. After this, a job offer for the open position is *created*. Then the offer is *mailed* to the newspaper. Finally the job offer is *finished* (filing, notification of departments, etc.). The expected duration in days is displayed on top of each task. Additionally, as the newspapers special "Job & Career" edition appears only once a month (on each 15th) a corresponding fixed-date constraint (fdc) has been defined on the last activity *mail*, demanding that it must be finished on the 14th of a month. Currently, on Sunday May 8th 2005, Mr. Smith, employed in the companies personnel division, has two work items from two different JobPosting-processes in his worklist. For sake of simplicity, we assume that every day is a working day, Saturdays and Sundays included.

Scenario 1.a) According to the FIFO-policy, suggested by the worklist client, he starts to execute the first work item, the task *prepare* of process *JobPosting1*, which will presumably take 6 days. Although the execution of *prepare* starts immediately, the job offer will, according to the expected task durations, not appear in the May-issue, as the *mail*-task will presumably be finished on May 21st and the process will be finished at May 24th. Unfortunately, the decision to execute *prepare* first implicitly postpones the execution of his second work item, the task *create* of process *JobPosting2*, for 6 days, to May 14th. *create* can be finished on May 16th and the next step *mail* on May 17th. Thus the job offers of the second process will, presumably, also not appear in this month special issue.

Scenario 1.b) If Mr. Smith had chosen to execute the second work item first, the *mail*-task of process *JobPosting2* could have been finished on time, until May 11th. Thus the FIFO-policy unnecessarily delays the second process for 30 days. Although this decision postpones the first work item *prepare*, the *JobPosting1*



Worklist for Mr. Smith			Sunday, May 8th 2005
ID	Task	Process	Received
34848	prepare	JobPosting1	May 6th 2005
34856	create	JobPosting2	May 7th 2005

Figure 1. Scenario1: job posting workflow

process will not be delayed, as it will also end at May 24th (which is the same end date as in Scenario1). Due to the fixed-date constraint defined on *mail*, enough buffer time exists to compensate the postponement! The scenarios demonstrate that an intelligent and predictive selection of work items can help to decrease the turn-around times of processes.

Although different aspects of time and time management have been investigated extensively in related research areas such as project management or job-scheduling none of the proposed techniques are suitable for time management in workflows, mainly due to the complex issues involved in workflow modelling and instantiation [**?**, 10]. Nevertheless several publications can be found in the area of workflow time management, e.g. [3, 6, 7, 9]. They propose the calculation of time plans which define execution intervals for each activity in a workflow, such that deadlines will not be violated. As far as we know no publication exists which examines the relation between postponement and delay.

In this paper we introduce the basics of a method which exploits knowledge about the workflow structure and time properties, to assists workflow participants when deciding which work item to handle next, in order to decrease turnaround times, increase throughput and avoid time-related escalations, by providing information about the delay to expect when selecting or postponing tasks.

2 Time properties

In addition to the workflows control-structure certain time properties have to be defined during the modelling process. Each node N is augmented with the expected *duration* $N.d$ in an arbitrary time unit like hours or days. Control nodes (like start end, splits and joins) usually have a duration of 0. The workflows overall execution duration is limited by a workflow *deadline* δ, which is defined as a time value relative to the start of the workflow. Additionally the execution of a node can be constrained to certain dates by a fixed-date constraint $fdc(N, F)$, expressing that the end of node N can only occur on certain dates specified by the *fixed-date object* F, where $F.valid(Date)$ returns true if the arbitrary date is valid for F. $F.next(Date)$ and $F.prev(Date)$ return the next and previous valid dates after $Date$. Assume that a fixed-date object f may for example be defined as a list of valid dates $f =$ *(12th of March, 25th of September)* or as an expression like $f =$ *every 3rd sunday starting with 6th of September.*

Based on the workflow structure and time properties, it is possible to calculate the *earliest possible end (EPE)* for each node, relative to the start time of the workflow. It depicts the earliest possible point in time an activity may end, defined by the sum of durations of preceding nodes. The EPE of a node is basically the sum of all durations of nodes executed before it, starting from the current node Cur and at current time now. The EPE of each node is calculated in a forward pass, starting from the current node. Consider the workflow from scenario 1 with $Cur = prepare$ and $now = may8$ (depicting the current time 8th of May 2005): $prepare.epe = may14$, $validate.epe = may19$, $create.epe = may21$ and $mail.epe = may22$. According to the fixed-date constraint defined on *mail*, the EPE must be shifted to the next valid date adhering to $fdc(mail$,14th of a month$)$, yielding $mail.epe = jun14$. Then we calculate $finish.epe = jun17$. Finally, as control nodes like End have a duration of 0, the earliest possible end of the workflow is is equal $End.epe = jun17$.

3 Postponement and Delay

A *postponement* designates the shift of the execution of an activity, which is ready to be executed by a participant, to a later point in time. A *delay* is generated if due to a postponement the earliest possible end of the workflow (= $End.epe$) would be increased. In our running example with $start = now = may8$ (start depicts the start time of the workflow and now the current time) and $Cur = prepare$ we additionally define an overall workflow deadline of $\delta = 90$, therefore the workflow (or the node End) must be finished until $start + \delta = aug6$. The examination of dependencies between delay and postponement must start at the last node. One can see, that the workflow can not end before $End.epe = jun17$ and must not end after $now + \delta = aug6$. Thus, the following can be stated (for End): if it is not postponed it will not be delayed and end at $jun17$, if it is postponed by 1 day it will be delayed by 1 day and end at $jun18$, and so on. Finally we can state, that if it is postponed by 50 days it will be delayed by 50 days and end at $aug6$. A further postponement will lead to a violation of the workflow deadline $now + \delta = aug6$.

Based on this knowledge it is possible to de-

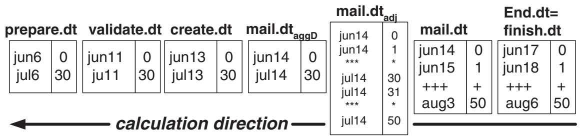


Figure 2. Delay tables for Scenario1

fine a relation between a *delay deadline (d)* and a *delay time (t)*, where t is the delay, if the end of the node is postponed to d. This relation is captured in the *delay table (DT)*. E.g. the DT of *End* is *End.dt*={*(jun17,0),(jun18,1),(jun19,2), +++,(aug6,50)*}. The *+++* between two tuples (d_1, t_1),*+++*,(d_n, t_n) symbolize that the set holds additional tuples (d_i, t_i), such that $d_i = d_{i-1} + 1$ and $t_i = t_{i-1} + 1$ for $1 > i > n$.

To express queries on DTs we define the operation *selectT*, which yields the delay time for a given delay deadline, e.g. *selectT(End.dt, jun18) = 1* states that the workflow will be delayed by 1 day if the node *End* ends at $jun18$. If no tuple with the requested delay deadline exists, the delay time of the tuple with the next greater delay deadline is selected. The selection of a point in time greater then the maximum delay time in a DT returns a result of ∞. This expresses the possibility of workflow deadline violation: e.g. *selectT(End.dt, sep12)* $= \infty$ states that if the workflow ends at $sep21$ the deadline will be violated.

4 Calculation of Delay Tables

4.1 Sequences

The DT of each node is calculated in a backward pass, starting from the end node, which must be initialized as stated above. For nodes, which are to be executed in a sequence, the DT of a predecessor node is calculated by applying an operation which subtracts the duration of the successor from each delay deadline in the successors DT, yielding *finish.dt = End.dt - End.d* and *mail.dt = finish.dt - finish.d* and so on (see also Fig. 2). As a fixed-date constraint *fdc(mail,f)* exists, the end of *mail* may only occur on valid dates, defined by the fixed-date object *f=14th of a month.* Thus the delay deadlines in *mail.dt* must be adjusted using the *previous* function of the fixed-date object. The operation yields $mail.dt_{adj}$ = {*(jun14,0),(jun14,1),***, (july14,30), (july14,31),***, (july14,50)*}. The ***** between two tuples (d_1, t_1), *****,(d_n, t_n) symbolize that the set holds additional tuples (d_i, t_i), such that $d_i = d_1$ and $t_i = t_{i-1} + 1$ for $1 > i > n$. As one can see that this operation may result in a DT with multiple identical delay deadlines (but different delay times). As exactly one delay deadline must unambiguously determine a delay time, all tuples with identical delay deadlines must be removed, but the one with smallest delay time, yielding $mail.dt_{aggD} = \{(jun14, 0), (jul14, 30)\}$. This is interpreted as: if $mail$ ends at $jun14$ or before, the delay will be 0; if it ends after $jun14$ and before or exactly at $july14$ the delay will be 30. The next possible end, according to the fixed-date constraint, is $aug14$. But if $mail$ would end at $aug14$, the workflow deadline would be violated. Thus the last allowed delay deadline (or end time) is *jul14*. The DTs of the remaining nodes are calculated using the subtraction operation.

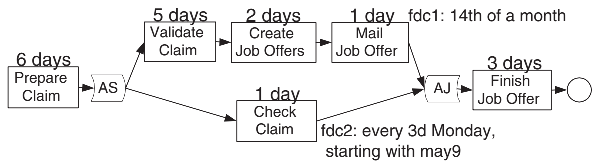


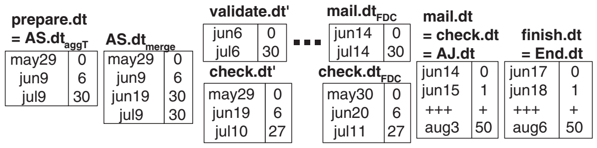
Figure 3. Scenario2: Parallel execution

4.2 Parallel Execution

Workflows may also contain control nodes of type *and-split* and *and-join.* Routes after an *and-split* will be processed in parallel and will be synchronized at the according *and-join*, which does not proceed until all predecessor nodes are finished. We adapt the example, as visualized in Fig. 3: The company decides that the management's staff unit must be informed about every claim. Additionally a fixed-date constraint is attached to the new activity *check* (which takes 1 day), as the staff unit team only meets every 3d monday (starting with may 9th) to discuss and check new claims. Assume that the semantics of *finish* is now: finally the job offer **and** the claim are finished together (filing, notification of departments, etc.). Note that it is assumed that control nodes have a duration of 0.

Again *Cur=prepare, now=may8* and *δ=90.* At first the EPEs must be calculated: *Start.epe=may8, prepare.epe=may14, AS.epe=may14, validate.epe=may19, create.epe=may21, mail.epe=jun14, check=may30, AJ.epe=jun14, finish.epe=jun17* and *End.epe=jun17.* The EPE of the new activity *check* has been adjusted to the next valid date $may30$ (= $may9$ + 3weeks) according to its fixed-date constraint. And, as the execution will not proceed at the and-joins until all predecessors are finished, the EPE of the and-split AJ is equal to the maximum EPE of its predecessors (*mail* and *check*), which is *mail.epe.*

The calculation of DTs again starts with the initialization of the last node (based on *End.epe* and *start* + δ), followed by the subsequent calculation of $finish.dt$ and *AJ.dt* (see also Fig.4). The DTs of nodes which are predecessors of an and-split are calculated



prepare.dt = $AS.dt_{aggT}$

may29	0
jun9	6
jul9	30

$AS.dt_{merge}$

may29	0
jun9	6
jun19	30
jul9	30

validate.dt'

jun6	0
jul6	30

check.dt'

may29	0
jun19	6
jul10	27

...

$mail.dt_{FDC}$

jun14	0
jul14	30

$check.dt_{FDC}$

may30	0
jun20	6
jul11	27

mail.dt = check.dt = AJ.dt

jun14	0
jun15	1
+++	+
aug3	50

finish.dt = End.dt

jun17	0
jun18	1
+++	+
aug6	50

Figure 4. Delay tables for Scenario2

like nodes in a sequence: $mail.dt = AJ.dt - AJ.d$ and $check.dt = AJ.dt - AJ.d$. Since $AJ.d{=}0$, their DTs are equal to $AJ.dt$. For the upper parallel route we refer to the calculations of Scenario1, as the DTs are equal. For the single node *check* of the second route we have to apply its fixed-date constraint *fdc(check, every 3d monday starting with may9)*, resulting in $check.dt_{fdc}$ (to save some space we did not show the intermediate results $check.dt_{aggD}$ and $check.dt_{adjust}$).

Before combining DTs of and-split successors, their durations must be subtracted (resulting in an intermediate dt'): *validate.dt'*={*(jun6,0),(jul6,30)*} and *check.dt'* = {*(may29,0),(jun19,6),(jul10,27)*}. At the and-split the DTs of all successors must be merged. The merge-operation selects the greatest delay time (as all routes are processed in parallel) for every possible delay deadline, which is for the running example accomplished as follows: First a set D which holds all delay deadlines of both sets is generated, which is for our example $D{=}\{may29, jun6, jun19, jul6, jul10\}$. Then for every delay deadline in D the *selectT*-operation is applied on both DTs and the maximum of the results is selected, which yields 0 for $may29$, 6 for $jun6$, 30 for $jun19$, 30 for $jul6$ and ∞ for $jul10$: $AS.dt_{merge}$ = {*(may29,0),(jun6,6),(jun19,30), (jul6,30)*}. The delay deadline $jul10$ has been filtered out as ∞ indicates an upcoming deadline violation. As one delay deadline must refer to exactly one delay time, it is necessary to remove the tuple with the earlier delay deadline resulting in $AS.dt_{aggT}$ = {*(may29,0),(jun9,6),(jul9,30)*}. And finally, due to $AS.d = 0$, the DT of the current activity *prepare* is $prepare.dt = AS.dt_{aggT} - 0$.

4.3 Conditional Execution: A Problem Statement

The workflow in Fig. 3 is rather simplistic as one fundamental concept has not been addressed so far: conditional execution. At an *or-split* exactly one route out of many will be selected according to a run-time evaluated condition. The corresponding *or-join* proceeds if one predecessor node is finished (the one on the route selected at the or-split). This raises some problems, e.g.: the EPE of an or-join can not be determined unambiguously, as it depends on the path which has been selected at the or-split. Which one this will be, is unknown at a current node Cur. To tackle this problem we introduced the concept of probabilistic delay tables along with detailed definitions and algorithms partially presented in this paper (see technical report [4]).

5 Conclusions and Future Work

We introduced the basics for a method to calculate delay tables which contain information about the delay to expect if a workflow participant postpones the end of a task in his worklist to a given date. This information may be used to provide additional selection criteria to the participant, e.g. tasks where the delay time does not change if postponed for some days or even weeks, or tasks where a deadline miss is very likely to occur. This will assist participants to select work items in order to decrease turnaround times, increase throughput avoid time-related escalations and subsequently save costs.

Currently we proceed with our investigations on the integration of probabilistic information about workflow execution and to find optimization algorithms for automatically presorted worklists. Furthermore we plan to investigate how to adapt our algorithms for non full-blocked workflows, which for instance allow the definition of arbitrary cycles.

References

[1] G. Baggio and J. Wainer and C. A. Ellis. Applying Scheduling Techniques to Minimize the Number of Late Jobs in Workflow Systems. In *Proc. of the 2004 ACM Symposium on Applied Computing (SAC)*. ACM Press, 2004.

[2] C. Bettini, X. X. Wang, and S. Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3), May 2002.

[3] C. Bettini, X. X. Wang, and S. Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3), May 2002.

[4] M. Bierbaumer, J. Eder and H. Pichler. Probabilistic Delay Times for Workflow Systems. Technical report, Universität Klagenfurt, Institut für Informatik Systeme, 2005.

[5] C. Bussler. Workflow Instance Scheduling with Project Management Tools. In *9th Workshop DEXA'98*. IEEE Computer Society Press, 1998.

[6] C. Combi and G. Pozzi. Temporal conceptual modelling of workflows. In *Proc. of the Int. Conf. on Conceptual Modeling (ER 2003)*. LNCS 2813. Springer, 2003.

[7] J. Eder and E. Panagos. Managing Time in Workflow Systems. In *Workflow Handbook 2001.* Future Strategies INC. in association with Workflow Management Coalition, 2000.

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

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

[10] S.Sadiq, O. Marjanovic, and M. E. Orlowska. Managing Change and Time in Dynamic Workflow Processes. In *International Journal of Cooperative Information Systems.* Vol. 9, Nos. 1 & 2. March - June 2000.