Escalations in Workflow Management Systems

E. Panagos and **M. Rabinovich**
AT&T Labs – Research
{thimios, misha}@research.att.com

Abstract

Workflow management systems (WFMSs) have been used to support the modeling, execution, and monitoring of business processes. Business processes consist of multiple activities, and their enactment is carried out by human agents and software systems. Typically, business processes and the activities constituting them have deadlines. When an activity misses its deadline, special actions may be triggered, referred to as *escalation*. Escalations affect business processes, and they may even lead to the abortion of some of them. Consequently, escalations may entail a high cost to an organization. In this paper, we present on-going research addressing workflow escalations. Our goal is twofold: (a) minimize the number of escalations during process execution and (b) reduce the cost associated with escalations when they cannot be avoided.

1 Introduction

Workflow management systems (WFMSs) are increasingly used by organizations to streamline, automate, and manage their business processes. These systems provide tools to support the modeling of business processes at a conceptual level, coordinate the execution of component activities according to the model, monitor the progress of business processes, and report "important statistics" of both the business processes and the systems involved in the execution of them [3, 8].

WFMSs are based on the concept of a *workflow*, which is an abstraction of a business process. A workflow consists of *activities*, which correspond to process steps, and *agents* that execute these activities. The workflow specification describes the activities constituting the workflow and the (partial) order in which these activities must be executed. Activities may have deadlines that determine the maximum allowable execution time for each of them. When an activity misses its deadline, the workflow model may specify that a special activity, referred to as *escalation*, be triggered automatically. The effects of an escalation may depend on the semantics of the activity that missed its deadline and, often, human intervention is required to proceed.

In general, the effects of an escalation may be one of the following: (a) the activity that triggered the escalation is restarted, (b) a new activity is executed and the activity that triggered the escalation resumes execution, (c) a new activity replaces the one that triggered the escalation, or (d) the business process is affected, and it is either aborted as a whole or some of the executed activities are compensated. In all cases, invoking escalation results in an increased cost for a business process due to the additional activities that have to be executed, or because completed work is rolled back, or because intervention of highly-paid workers is required. Therefore, it is desirable to reduce the number of workflow executions that result in escalations.

In this paper, we address the problem of reducing both the number of workflow executions that result in escalations and the cost associated with escalations. The main observation behind our techniques is that as long as we guarantee that activities are given at least as much time as originally assigned by the business analyst, we can delay the invocation of escalations for some of them by giving them more time to complete normally. To achieve this, we note that deadlines are usually assigned to activities based on the estimated execution times of these activities and on the need to meet the overall deadline of the business process. On the other hand, the *actual* time required for a particular activity to complete varies from one instance to the next due to variations of load, work conditions, etc. Consequently, if an activity in a given workflow execution finishes faster than its estimated execution time, the rest of the activities can be given extra time before escalation is invoked. This can be accomplished by extending their deadlines using the available slack time.

Although our work shares similar goals with research efforts in real-time systems (namely, that of reducing the number of tasks that miss their deadlines), the focus of our work is completely different. Real-time systems attempt to minimize the number of missed deadlines by optimizing scheduling of tasks so that each task better utilizes its allowable execution time, which is assumed to be fixed. Our work, on the other hand, attempts to minimize the missed deadlines by modifying the deadlines themselves, without changing the scheduling policy. The key observation is that, in the workflow application domain, one can do that without changing the semantics of the business process. In other words, our deadline adjustment is transparent to the users.

The remainder of the paper is organized as follows. Section 2 introduces the workflow model. Section 3 presents several algorithms that adjust the deadlines of the remaining activities on-line. Section 4 compares our work with related research. Finally, Section 5 summarizes the work presented in this paper and addresses on-going research.

2 Workflow Process Model

We consider a business process (i.e., a workflow) consisting of n activities $T_1, \ldots, T_n$ with a partial order defined by a *successor function* [2] which associates every activity T_i with a set of activities, named *successors*, that are executed after T_i. Each successor activity is executed unconditionally or only when a condition associated with it evaluates to `true`. Moreover, we assume that repeated execution of activities is not allowed, and there is only one activity that marks the completion of the business process.

Although the class of workflows supported by our model is a proper subset of the class of workflows supported by the Workflow Management Coalition model [4], it is particularly important because it frequently appears as a building block for more complex workflows. Therefore, workflows containing groups of activities that follow our model can benefit from our algorithms by applying them to these groups. Moreover, while our algorithms are not applicable to workflows that do not comply with our model, these workflows are still allowed in the system.

In order to simplify our presentation, we further restrict the above model. In particular, we assume that a business process consists of m sequential activities $T_1, \ldots, T_m$. Each activity T_i may consist of n_i sub-activities ${T_i}^1, \ldots, {T_i}^{n_i}$ to be executed in parallel. Each sub-activity may be executed either conditionally or unconditionally, and it cannot be decomposed any further. In addition, we make the following assumptions regarding the workflow management system and the information that is available about the activities constituting the business process.

- Scheduling of the activities that are ready for execution is not based on the deadlines assigned to them[1]. To the best of our knowledge, the above assumption is true for all commercial workflow products and existing research prototypes.
- Each activity has a deadline, which specifies the allowable execution time for the activity. We assume that deadlines are assigned by a business analyst based on the activities' estimated/predicted execution times and on the need to meet the overall business process deadline.

In addition, some of our algorithms assume that the following information is available.

- An estimate of the expected execution time for each activity. This estimate corresponds to the time it takes the agent, which will execute the activity, to complete the activity after the activity is submitted for execution. Most existing WFMSs maintain such estimates based on the accumulated history of prior executions.

[1] Under the *earliest deadline first* scheduling policy, the scheduling priority of a task decreases as the task's deadline increases. Consequently, our algorithms may increase the number of escalations because they extend deadlines.

- An escalation cost associated with each activity[2]. Escalation costs are assigned by business analysts based on the effects of escalation should the activity miss its deadline. We assume that the escalation cost is an integer number greater or equal to one. The higher the number the more costly the escalation procedure is.

3 Dynamic Deadline Adjustment

In this section, we present a subset of the algorithms we are currently developing for dynamic deadline adjustment of workflows that conform to the model and assumptions presented in the previous section. Associated with an activity T are the attributes denoted by the following functions (similar to the notation used in [6, 7]).

$$\begin{array}{lll} \mathrm{dl}(T) & = & \text{deadline assigned to } T. \\ \mathrm{ex}(T) & = & \text{estimated execution time of } T \\ \mathrm{sl}(T) & = & \text{slack of } T \text{ after its execution} \\ \mathrm{ce}(T) & = & \text{cost of escalation assigned to } T \end{array}$$

The slack of T is computed after T finishes execution, and it is equal to the difference between T's deadline and T's actual execution time.

With each business process instance we associate a variable *Slack* that corresponds to the available slack time for the process instance at any point in time during its execution. Before the very first activity of the process is submitted for execution, the value of *Slack* is set to 0. Before an activity T_k is submitted for execution, its deadline is adjusted by adding to it a portion of *Slack*, which is determined by the algorithms presented below, and the new value of *Slack* becomes equal to its previous value minus the portion assigned to T_k's deadline. When T_k finishes execution, the value of *Slack* is incremented by $sl(T_k)$.

Total Slack (TSL): Without any knowledge about the estimated execution times or escalation costs of the activities, we adjust the deadline of the activity that is going to be executed next by adding the available slack time to it.

$$dl(T_k) = dl(T_k) + Slack$$

However, assigning the whole slack to the next activity may sometimes be suboptimal. By using up the entire slack in activity T_k, we may avoid escalation for T_k at the expense of increasing the risk of an even more expensive escalation for some later activity. In addition, the fact that an activity is not finished within a certain time period is often an indication of a long-lasting problem and, thus, delaying escalation beyond a certain point may not be cost effective.

Proportional Execution (PEX): Knowing the estimated execution times of the activities constituting the business process, we distribute the available slack in proportion to these times. For parallel activities, the estimated execution time is set to the maximum of the estimated execution times of their sub-activities. If the next activity to be executed is T_k, the following formula is used.

$$dl(T_k) = dl(T_k) + Slack * \frac{ex(T_k)}{\sum_{j=k}^{m} ex(T_j)}$$

Proportional Escalation (PES): The proportional escalation mechanism discriminates among activities with different escalation costs. In particular, activities with higher escalation costs are assigned larger portion of the available slack time. For parallel activities, the escalation cost is set to the maximum of the escalation costs of their sub-activities. If the next activity to be executed is T_k, the following formula is used.

$$dl(T_k) = dl(T_k) + Slack * \frac{ce(T_k)}{\sum_{j=k}^{m} ce(T_j)}$$

[2]Typically, WFMSs allow user-defined attributes to be associated with activities and, thus, one of these attributes could be used for escalation costs.

4 Related Work

Dynamic deadline adjustment is principally different from dynamic modification of workflows supported by some existing workflow products and research prototypes. The latter is done to reflect changes in the model of the business process or a particular instance of the process. In contrast, our goal is to minimize the operation cost *without* modifying the business process model. In this, it is somewhat similar to scheduling in real-time systems [9, 1, 5]. However, real-time systems use deadlines for scheduling system components such as CPU and I/O. We view scheduling and dynamic deadline adjustment as complimentary mechanisms. We will report separately on more sophisticated scheduling policies as well as on inter-dependencies of policies for scheduling and deadline adjustment.

Our work is related to the work described in [10]. In [10], the authors present priority-driven CPU scheduling algorithms for transactional workflows. Each workflow process consists of several sequential tasks. Each task is an ACID transaction having an average response time goal. The assignment of priorities is based on the performance of the tasks relative to their original response time goals. In contrast to these algorithms, our work does not concentrate on CPU scheduling. In addition, our algorithms are not restricted to transactional workflows, and they allow both sequential, conditional, and parallel execution of tasks.

In [6, 7], the authors study the problem of how the deadline of a real-time activity is automatically translated to the deadlines of all sequential and parallel sub-tasks constituting the activity. Each sub-task deadline is assigned just before the sub-task is submitted for execution, and the algorithms for deadline assignment assume that the *earliest deadline first* scheduling policy is used. Similar to our work, the goal of the proposed algorithms is that of minimizing the number of missed deadlines. In contrast to these algorithms, our work assumes that all sub-tasks have been assigned deadlines before the activity is submitted for execution and, thus, we focus on adjusting sub-task deadlines on-line rather than assigning deadlines. In addition, [6, 7] assume a soft real-time environment where sub-tasks may miss their deadlines without triggering other sub-tasks. In our environment, however, escalations caused by missed deadlines may result in the execution of new activities, and our algorithms take into account the cost associated with these escalations.

In active, real-time databases transactions may trigger new transactions during their execution. The triggered transactions are executed immediately or after the original transaction finishes execution. In [11], the authors present algorithms that try to minimize the number of triggering transactions that miss their deadlines under a priority-driven scheduling policy. The proposed algorithms assign priorities to the triggered transactions, and they dynamically reassign the priorities of the triggering transactions. In contrast, our algorithms modify the deadlines assigned to activities without altering their priorities, if any. Consequently, our work could be complemented with the scheduling algorithms proposed in [11].

5 Summary and Future Work

Workflow processes consist of several activities which are executed by human and software agents. In many cases, activities have deadlines and escalation takes place when a deadline is missed. Minimizing the number of escalations during the enactment of business processes is highly desirable because escalations usually result in high cost to an organization.

In this paper, we have presented several techniques that aim at reducing the number of escalations during the execution of business processes. Our algorithms adjust the deadlines of the activities that remain to be executed by distributing any available slack time to them. Currently, we are extending our algorithms so that: (a) the load of the agents responsible for the enactment of activities is taken into account, and (b) empirical knowledge regarding the conditional execution of activities is used for distributing slack to the various branches. Furthermore, we are planning to evaluate our algorithms by implementing them on top of a commercial WFMS.

Although the techniques presented in this paper try to reduce the number of escalations, escalations cannot be always eliminated (e.g., when the resources required for the enactment of an activity are not available for a time period greater than the overall deadline of the business process). In such cases, it may be beneficial to force escalation at an *early stage* during the process execution to reduce the escalation cost. We are in the process of developing algorithms to decide whether and when to force escalation by exploiting knowledge about the costs of the escalation procedures that may be invoked during the process enactment as well as information regarding the status (load, availability, etc.) of the agents participating in the execution of the process.

Another research avenue we are currently pursuing is that of providing mechanisms for automatically assigning deadlines to activities when only an end-to-end deadline is known about the workflow process. Although having a single deadline is simple, it is not adequate for monitoring the progress of individual activities and detecting abnormal conditions and potential problems early enough. However, the automatic assignment of deadlines to individual activities is not trivial. The main obstacle is the fact that activities may be executed repeatedly, with the number of iterations determined only at run-time. In addition, activities may be executed conditionally based on information that becomes available at run-time. Finally, activities may have different resource requirements and escalation costs, which should also be taken into account.

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] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *Proceeding of the ER'96 International Conference*, Berlin, October 1996.

[3] 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.

[4] The Workflow Management Coalition Group. A workflow management coalition specification. Technical report, The Workflow Management Coalition, November 1994. http://www.aiai.ed.ac.uk/WfMC/.

[5] J. Huang, J.A. Stankovic, D. Towsley, and K. Ramamritham. Experimental evaluation of real-time transaction processing. In *Proceeding of the 10th Real-Time Systems Symposium*, December 1989.

[6] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 428–437, 1993.

[7] B. Kao and H. Garcia-Molina. Subtask deadline assignment for complex distributed soft real-time tasks. Technical Report 93-1491, Stanford University, 1993.

[8] N. Krishnakumar and A. Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2):1–33, 1995.

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

[10] M. Marazakis and C. Nikolaou. Towards adaptive scheduling of tasks in transactional workflows. In *Winter Simulation Conference*, Washington D.C., 1995.

[11] R.M. Sivasankaran, J.A. Stankovic, B. Purimetla D. Towsley, and K. Ramamritham. Priority assignment in real-time active databases. *The VLDB Journal*, 5(1):19–34, 1996.