

# Time Management for Mission-Critical Workflow Systems<sup>1</sup>

Mariusz Momotko, Lilianna Zalewska

Rodan Systems,  
ul. Pulawska 465  
02-844 Warszawa, Poland  
{Mariusz.Momotko, Lilianna.Zalewska}@rodan.pl  
<http://www.rodan.pl>

**Abstract.** There is a relatively vast research on time management in workflow systems. We found work of Eder et al. [5, 6] especially feasible due to its simple yet powerful time model. This technique was implemented into our commercial workflow engine, namely OfficeObjects<sup>®</sup>WorkFlow. Its implementation, while proved to be sufficient in less demanding applications, revealed some drawbacks in mission-critical workflow systems. In this paper we suggest that the algorithm to verify activity deadlines should be compliant with more advanced activity instance behavioural model. We define such model and provide the enhanced algorithm. We also underline the benefits of exposing information about time constraints violation at the performer's level and consider two types of such information: to indicate whether activity is delayed and to check if it delays the whole process (i.e. is critical for the process). The latter type is used for better prioritising of the delayed activities.

## 1 Introduction

Nowadays organisations to be competitive and to play the leading role on the market need innovative, flexible and efficient business process management systems (BPM systems). One of the key requirements for such systems is the ability to manage time, especially to model time constraints, control them during process execution and handle time errors. Appropriate time management may help process performers in executing their tasks on time, early detection of time constraints violation, and taking right decisions to reduce existing delays. In addition, time related data recorded during process execution may then be used to estimate average values of some temporal types such as activity duration, and activity waiting time.

So far one of the most promising class of BPM systems are workflow management systems (WfM systems). WfM systems enable business processes to be defined, executed, monitored and analysed. The main goal of a WfM system is to assure that a given process activity will be executed by appropriate performer, at right time and on correct data.

---

<sup>1</sup> This work was supported by the European Commission project ICONS, IST-2001-32429

Despite rich functionality offered by WfM systems, supported time management seems to be insufficient. In particular, existing time management functionality is restricted to process simulation (to estimate duration of individual activities, identify potential bottlenecks, etc.), assignment of deadlines to activities, and triggering of escalation rules. Such features as consistency of time constraints and the side effects of time constraints violations are rather neglected [5, 6]. Moreover, none of the currently available technique found in other time management applications (e.g. project management, temporal databases, or artificial intelligence) is suitable and directly applicable for time management in WfM systems [9].

Recently, the workflow research community has put substantial effort to define better techniques for managing time in WfM systems. In [5, 6, 7] the authors proposed a technique to model (temporal types such as duration and deadline), compute (timed graph construction) and verify time constraints (run-time deadline calculation). The proposed workflow model is able to express sequential as well as parallel routing. For parallel routing alternative, conditional and unconditional routing elements are supported. In addition, also optional activities are expressed. Construction of the timed graph consists of calculated E-values and L-values for every process activity. These values are then used during process execution to indicate the timing status of a given process instance. The other concepts for time management have been elaborated in the ADEPT project [4]. In the ADEPT's approach for every activity four additional attributes corresponding to earliest and latest starting and finishing time are defined. To express more complex time dependencies between activities, so called time edges are used. The ADEPT system monitors process execution with respect to the calculated time constraints and informs performers if deadlines are going to be missed. Yet, another approach for time modelling is presented in [9]. This approach is based on the idea of two-level conceptual workflow modelling: the high-level or control flow level, and the operational-level model. These two models are constructed to be semantically equivalent. The latter model is used for time modelling including modelling temporal constraints and verification of their temporal consistency.

In the ICONS project we found the work of Eder et al. especially feasible due to its simple, coherent and powerful time model. The presented algorithms were implemented into a commercial workflow engine, namely, OfficeObjects®WorkFlow. Its implementation, while proved to be sufficient for less demanding workflow applications (small number of tasks to perform, not too rigorous time constraints), revealed some drawbacks for mission-critical WfM systems (enormous number of tasks to perform, rigorous time constraints). Yet, we found that the mentioned technique provides the performers not only with information whether a given activity is delayed but also if it delays the whole process. In some sense this information indicates if the activity belongs to the critical path of the process. In our opinion, this information may help both the process owner and process performers deciding which activity must be speed-up in the first, and which, despite their delay, may be executed in the second order (i.e. tasks prioritising). Finally, we realised that time constraint calculation as well as exposing time information should be based on process/activity behavioural models.

The structure of the document is as follows. In the first stage we specify new requirements for time management revealed during implementation. The described

requirements are provided with some examples of ‘problematic’ situations and indications where information about criticality of an activity may be useful. Based on the specified requirements, in the next stage we define appropriate behavioural model for activity instance. This model is an extension of those presented in workflow standards such as [1, 12]. Then we provide the updated algorithm for verification of deadline calculation. This algorithm is based on the defined behavioural model. Finally, we summarise our work and discuss possible areas of future extensions.

## **2 Implementation – a source of new requirements**

In the ICONS project we chose the time management technique proposed in [5, 6]. We implemented it in our OfficeObjects<sup>®</sup> WorkFlow system and deployed at our two selected customers. The first of these customers was characterised by significant number of executed processes (about 200 processes started every day) and quite rigorous time constraints. The half of the processes was short time processes and lasted no more than two working days. The rest of them lasted about ten working days. The second customer possessed three simple processes that were executed rarely (about several processes a day).

During three-month test exploitation, we were collecting carefully the results for application of this technique. The results confirmed that the technique is appropriate for light-loaded workflow systems with not too many tasks to execute (i.e. short work lists) and the expected activity durations are not too rigorous. For mission-critical systems, however, the results showed that the assumed activity behavioural model was too simple and needed to be extended. Then such extended model should have been applied in runtime deadline calculation. Also, we observed that this technique gives additional information about the time constraint violations at the performer’s or activity level. Again, in our opinion this information is important in mission-critical WfM systems. In such systems the delayed activities occur quite often and the performers have to take decision which delayed activities should be executed first, because, for example, they delay the whole process. On the basis of the collected results we defined new requirements for this time management technique and underlined some its important aspects that have not been considered in the mentioned articles. The most important requirements and issues are described in the next sections.

### **2.1 Waiting Time**

In the presented technique, verification of the calculated activity deadlines is based on a very simple model of activity behaviour. This model includes two states, namely ‘Open’ and ‘Closed’. The former state means that a given activity has been created, assigned to the performer and its execution started immediately after its creation. The latter state means that the activity has been closed (normally or abnormally). As was mentioned earlier, such model is sufficient for less demanding WfM systems. In these systems work lists include not too many work items and these items are performed almost immediately after they appear on a given work list. In other words, time when a

work item is waiting on the performer's work list is much shorter than activity execution time (see **Fig. 1**, case a). These temporal types have already been defined in the WfMC standard (see [12]) and called *waiting time* and *working time* respectively.

However, in real, mission-critical WfM systems waiting time for activities may be significant and comparable to working time (see **Fig. 1**, case b). The process execution metrics we achieved at our heavy-loaded customer confirmed it. Yet, some reports indicate that waiting time may be even much longer than working time [8]. In some cases neglecting waiting in time verification may cause already expired calculated deadline not to be reported (because the activity has not been started and, according to the original model, it has not moved to the 'open' state yet).



**Fig. 1.** Different proportion between waiting time and working time

In order to take into consideration waiting time in the presented time management technique, the activity behavioural model needs to be extended of waiting state. This state means that a given activity (or work item) has already been created and assigned to the performers but it is still waiting on its work list to be executed. Then, on the base of this extended behavioural model, the verification algorithm has to be updated. This model and the updated algorithm are defined more precisely in the next chapters.

## 2.2 Precise Calculation of Working Time

Moreover, the algorithm for verification of the calculated activity deadlines assumes that a given activity is executed in one step (or as a 'one shot'). This assumption is appropriate for simple activities. In real processes, however, activities may be more complex. Usually, complex activities are performed in several steps. For instance, verification of a one hundred pages document that includes five chapters may be done in five steps; each step for each chapter.

Execution of activities in more than one step requires working time to be calculated as a sum of working time of all the activity steps rather than duration between starting execution of the first step and finishing the last one. Not considering activity steps duration may lead to inappropriate verification of expected duration (i.e. duration set explicitly by the process owner).



**Fig.2.** Different activity execution

In our opinion such feature should also be expressed in the activity behavioural model as two sub-states: 'running', 'not running' of the 'open' super-state. The former state means that a given activity is currently being executed. The latter state means

that the performer has already started to execute this activity but currently, it is not being executed.

### **2.3 Exposing Time Constraint Violation at the Performer's Level**

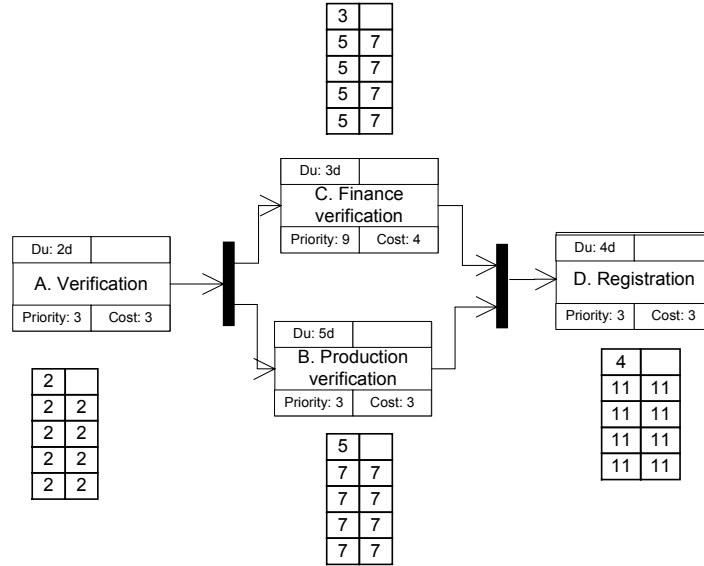
The presented time management technique allows two temporal types that can be defined at the activity level, namely duration and deadline. Activity duration means the maximum allowable time period between the activity starts (i.e. entering the 'open' state) and the activity ends (i.e. entering the 'closed' state). Activity deadline means maximum allowable time when the activity has to be finished (i.e. to enter the 'closed' state) with respect to the process creation time (e.g. verification of the order must finish no later than five hours after making the order).

In the presented technique, during process definition the WfM system calculates E and L-values for every activity within the process. E-values correspond to earliest possible times when activity has to be finished, while L-values correspond to the latest possible time (see [6]).

During process execution the WfM system checks whether one of the calculated L-values is violated. If so, the WfM system informs the process owner or administrator that it may be necessary to drop some optional activities, or to choose faster alternatives or finish the remaining activities faster than expected. Since this information is valuable for the whole process, it is usually available for the process owners and hidden and unavailable for ordinary process performers. Usually performers operate via their work list. A work list includes all work items (or activities) that have been assigned to a given performer. In existing WfM systems, a work item also contains information whether it is delayed or not (i.e. whether one or more of its time constraints is violated).

During implementation of the time management technique we observe that it gives the performers even more precise information of activity delays. If for a given activity calculated  $L_b$  deadline is not satisfied, it means that this activity delays the whole process and, in some sense, belongs to the critical path of the process.

Moreover, it is possible that the activity is delayed (i.e. its duration already expired), but does not delay the process and thus, it is still a slack time for this activity.



**Fig.3.** Example workflow process

For instance, in the process presented in **Fig.3**, the activity C has explicit duration set to 3 days and calculated  $L_b$  values set to 7 days. If the activity A was performed in 2 days and the activity C has already been performed for 4 days, it means that the activity C is delayed but does not delay the whole process ( $L_b$  values are set to 7 days). Such a situation occurs because the activity C does not belong to the critical path of the process.

The consequence of this information is quite important for process performers. If an activity is delayed and delays the whole process, it should be executed first. If an activity is delayed but does not delay the process, it may be postponed a bit and executed in the second order (we assume that in both cases the original priority of the activities, before their delays, was the same). Such an approach may help performers with an enormous number of work items assigned, focusing on the most important tasks. This feature is referred to further in this document as *criticality behaviour*. The way to include this criticality behaviour within the behavioural model is described in the next chapter.

### 3 Activity Instance Behavioural Model

On the basis of the requirements for waiting time, calculation of working time and other requirements for process execution, we define an extended behavioural model for activity instance. We also propose to build a deadline verification algorithm upon this model. Such an approach should be more coherent with workflow standards (behavioural models defined in [1, 13]), easier to understand (some aspects such as multiple step execution are easier to define using the model) and also easier to implement (state machine). This model is able to express three types of behaviour, that is *operational*

*behaviour* related to activity processing operations, *timing behaviour* related to possible activity delays and *criticality behaviour* indicating whether a given activity instance delays the whole process. These kinds of behaviour do not depend (directly) on each other.

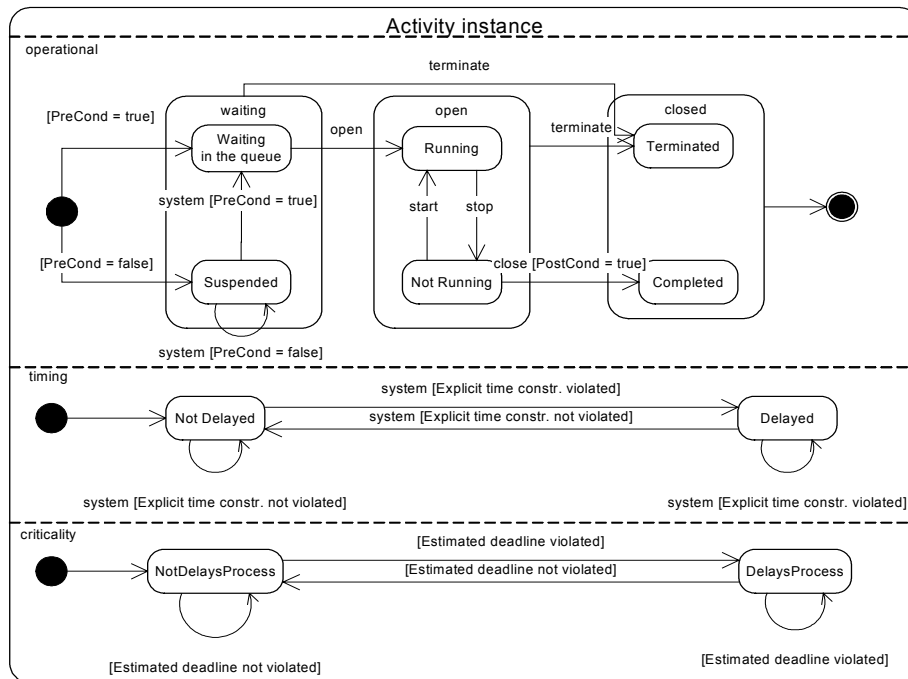


Fig. 4. Activity instance behavioural model

For **operational behaviour** when an activity instance is created, its pre-condition is checked. If it is satisfied, the activity is automatically set in the Waiting.Waiting in the queue state. Otherwise, it is set to Waiting.Suspend state. In the latter state it is checked periodically if the pre-condition is satisfied. If so, the activity changes its state to Waiting.Waiting in the queue. The activity being in this state is presented as a work item in the performer's work list. If the performer starts executing the work item, the activity changes its state to Open.Running. It remains in this state as long as the performer is executing the activity. Then the performer may finish execution of this activity or only postpone it. In the former case the activity changes its state to Closed.Completed if its post-condition is satisfied. If it is not, the instance remains in the current state. In the latter case it changes its state to Open.NotRunning. If the performer starts it again, it changes its states to Open.Running. Anytime when the activity is in Waiting.\* or Open.\* states it can be terminated. In this case it changes its state to Closed.Terminated.

For **timing behaviour** when the activity instance is created, it is automatically set to the NotDelayed state. If for this activity an explicit duration or deadline has been set, it is automatically assigned to it. If during process execution the explicitly set duration or deadline has been violated, the activity changes its state to Delayed. If in this state

the process owner adjusts the activity duration or deadline and none of them is violated, the activity instance changes its state to NotDelayed.

For the **criticality behaviour**, when the activity is created, it is automatically set to NotDelaysProcess state. Also estimation for duration and deadline is calculated. If during activity execution one of these implicit time constraints is violated, the activity delays process and changes its state to DelaysProcess. If in this state the process owner adjust the activity duration or deadline and after re-calculation of the estimated time constraints none of them is violated, the activity instance changes its state to NotDelaysProcess.

The UML state chart diagram for activity instance is presented in Figure 4. The meaning of the individual states is summarised below:

- Waiting.Waiting in the queue – the pre-condition for the activity instance has been satisfied, assigned to a performer and started successfully. However, so far, the work item related with this activity has not been opened by the performer.
- Waiting.Suspended – the pre-condition for activity instance has not been satisfied so far.
- Open.Running – work item has been taken by the performer and is executing.
- Open. Not Running –work item has been taken by the performer but is not currently executed.
- Closed.Completed – activity has been finished.
- Closed.Terminated – activity has been stopped (abnormally) due to error or user request.
- Not Delayed - activity instance deadline and duration (i.e. given explicitly by the process owner or implicitly by time deadline calculation) are still satisfied.
- Delayed - the activity instance deadline or duration (i.e. given explicitly by the process owner or implicitly by time deadline calculation) has already expired.
- Not delays process - execution of the activity instance does not influent deadline for the process.
- Delays process - execution of the activity instance influents deadline for the process and delays it.

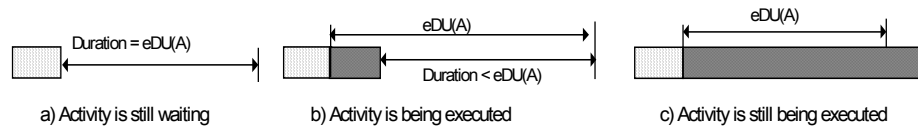
## 4 Updated Deadline Verification Algorithms

On the based of the activity behavioural model defined in the previous chapter we specify the updated algorithms to verify calculated deadlines. In addition to the time management terminology defined in [5, 6, 7], we introduce four symbols:

- $TS_{state}(A)$  - time when activity instance A entered a given state or a super-state. For instance,  $TS_{Open.*}$  means time when activity A entered the Open.\* super-state.
- $TF_{state}(A)$  - time when activity instance A left a given state or a super-state. For instance,  $TF_{Open.*}$  means time when activity A left the Open.\* super-state.
- $DU_{state}(A)$  – the total, current period of time which activity A has spent in a given state or super-state so far. For instance,  $DU_{Open.Running}$  means period of time when activity A spent so far in the Open.Running state.

- $eDU(A)$  – according to the WfMC’s terminology [12], it means expected period of time to perform activity. Usually, it is defined on the base of execution history as the average working time.

In the algorithm given in [6], verification of calculated deadlines depends on comparing the expression  $now + duration(A)$  to individual L-values. During implementation we found that it is very important to give the precise definition what duration means in this context. Following the WfMC’s terminology, duration in the above expression means estimated period of time required for completing a given activity. Such defined duration is not a constant value and decreases by elapsed working time. If the activity has been created, assigned to the performer but still not taken by him/her (i.e. still in the *Waiting.\** state), duration equals to the expected duration (i.e.  $eDU$ ). However, when the performer has already started the activity, the value of duration decreases by elapsed working time (i.e.  $DU_{Open..Running}$ ). Moreover, if the activity has already been performed longer than expected, the value of duration is zero.



**Fig. 5.** Activity instance behavioural model

The crucial point of the above calculations is that the value of duration depends on the current state of the activity instance. If activity is in the *Waiting.\** super-state, duration equals to the expected duration value (i.e.  $duration(a) = eDU(A)$ , Figure 5, case a). If activity is in the *Open.\** super-state – we have three possible situations:

- activity has just started ( $DU_{Open.*} = 0$ ) - this case is similar to the situation, when the activity is in the *Waiting.\** super state.
- activity is being performed for a while, but it’s execution time doesn’t exceed (Figure 5, case b) the expected duration – the value of duration equals to the expected duration minus period of time the activity has already spent in the *Open.Running* state:  $duration(A) = eDU(A) - DU_{Open..Running}(A)$
- actual activity duration time exceeds expected duration time (Figure 5, case c)– the value of duration equals to zero:  $duration(A) = 0$

## 5 Summary and Future Work

After defining the extended verification algorithms we implemented it once again in our OfficeObjects® WorkFlow system. In the next stage we deployed the system at one of the mentioned two our customers – the heavy-loaded one. In addition, in the new version we also applied the extended Business Process Modelling Notation [2] of process execution representation [11]. This extended notation enables process execution to be represented in similar way as process definition. In addition, it makes possi-

ble to represent the described behavioural activity instance model, also timing and criticality behaviour.

So far, after two-month deployment, we observe that the proposed extension increases the accuracy of the calculated deadline verification algorithm and gives the performer opportunity to better manage the work items that are already delayed. Moreover, introducing the new calculations into the algorithm did not reduce its performance.

As the future work we would like to enable this algorithm to cope with distributed workflow processes where the transferring time (i.e. time between finishing one activity and creating the next one) may be significant and should be taken into consideration. Also we would like to extend the existing deadline calculation algorithm and allows it to use workflow relevant data (variables) as deadline or duration values. Thus it could make the algorithm more dynamic. We believe that such approach may help solving the problem when time constraint values strictly depend on the data that are processed and, so far, need to be expressed at two levels of process modelling [9].

## References

- [1] Business Process Management Initiative; Business Process Modelling Language, 2002.
- [2] Business Process Management Initiative; Business Process Modelling Notation; working draft, version 0.9, Nov 2002.
- [3] C. Bussler, Workflow instance scheduling with Project management Tools. the 9<sup>th</sup> Workshop on Database and Expert Systems Applications, DEXA'98, Vienna, Austria, 1998.
- [4] P. Dadam, M. Reichert, and K. Kuhn. Clinical Workflows the Killer Application for Processoriented Information Systems, 4<sup>th</sup> International Conference on Business Information Systems, BIS'2000, Poznan, Poland, 2000.
- [5] Eder, J., Paganos, E., Managing Time in Workflow Systems, in Workflow Handbook 2001, Layna Fischer (Ed.), Future Strategies Inc., 2001, USA.
- [6] Eder, J.; Panagos, E., Pozewaunig, H., Rabinovich, M., Time Management in Workflow Systems, 3<sup>rd</sup> International Conference on Business Information System (BIS'99), pp. 265-280, 1999.
- [7] Eder, J.; Pozewaunig, H., Liebhart, W., ePERT: Extending PERT for Workflow Management Systems, 1<sup>st</sup> East-European Conference on Advances in Databases and Information Systems (ADBIS'97), 1997.
- [8] Koulopoulos, T., M., the Workflow Imperative, van Nostrand Reinhold, 1995.
- [9] O. Marjanovic, Methodological Considerations for Time Modelling in Workflows, 12<sup>th</sup> Australasian Conference on Information Systems, Australia, 2001.
- [10] O. Marjanovic, M. Orłowska, On modelling and verification of temporal constraints in production workflows. Knowledge and Information Systems 1(2), May 1999.
- [11] Momotko M., Nowicki, B., Visualisation of (Distributed) Process Execution based on BPMN, submitted to 14<sup>th</sup> Conference on Web Based Collaboration, Prague, Czech, 2003.
- [12] Workflow Management Coalition, Workflow terminology & glossary, WfMC-TC-1011 issue 3.0, Feb 1999.
- [13] Workflow Management Coalition, Workflow standard, Workflow process definition language – XML process definition language, WfMC-TC-1025 draft 0.03a, May 2001.