Task Scheduling for a Temporal Workflow Management System

Carlo Combi
Dipartimento di Informatica
Università degli Studi di Verona
strada le Grazie 15
I-37134 Verona, Italy
carlo.combi@univr.it

Giuseppe Pozzi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
p.za L. da Vinci 32
I-20133 Milano, Italy
giuseppe.pozzi@polimi.it

Abstract

A workflow management system (WfMS) supports the coordinated execution of simple activities (tasks), assigning them to human or automatic executors (agents), to achieve the goal defined for a business process. The workflow scheduler performs task assignment policies: temporal aspects, such as availability time and skills of agents, deadlines for task completion, constraints over timestamps for task activation, periodic constraints cannot be neglected. By this paper we propose a temporal workflow scheduler which pervasively considers all the temporalities affecting task assignment policies.

1 Introduction

Workflows are activities involving the coordinated execution of single activities (*task* or worktask) performed by different processing entities (*agent*, i.e. a person, a software system or both of them). The workflow specification describes the component tasks, their controlled and coordinated execution, the processing entities. Workflow management systems (*WfMS*) support the automatic execution of workflows. A WfMS provides [8]: the process model, i.e. the description (*schema*) of the managed business process, whose instances are known as *cases*; the information model, i.e. process-related data, like workflow historical data, data related to the process like the customer's name, and external data; the organizational model, i.e. the organization where cases are enacted by agents. All of these data are generally stored by a database management system (*DBMS*).

A WfMS manages information with associated temporal, such as the starting and ending timestamps of a task, the deadline for completing an activity, the lifespan of a task (from the scheduling of the task to its completion) and of a data item, the time interval between the execution of two different activities. Temporalities are to be dealt with in: (i) managing the change of a process schema [1]; (ii) expressing constraints in any process model [2]; (iii) defining the overall architecture of a temporal WfMS [3].

In this paper we focus on temporalities in the conceptual organizational model and in task assignment policies: a correct use of temporalities may improve the overall performances of the scheduling activity of a WfMS. To this regard, contributions from the field of generic scheduling and of other enterprize resource allocation and planning focusing on optimized algorithms and techniques are complementary to (and will be integrated with) our proposal. In the following, Section 2 surveys related work; Section 3 provides a brief overview of atemporal (timeless) and temporal organizational models; Section 4 describes the temporal information managed by a workflow scheduler during task assignment; Section 5 sketches out some final remarks.

2 Related Work

The literature presents several contributions about the management of time inside a WfMS. Eder et al. [4] specify the temporal properties for a task (namely, "activity node"), considering the duration, and the earliest finish and latest finish times, and extend the model by adding the best and worst case of execution to cope with possible conditional branches. Sadiq et al. in [9] briefly list three different types of constraints without going into details: *duration constraint*, defining the expected duration of every task, which can possibly be modified at case execution time according to some local variables or to the workflow history; *deadline constraint*, defining the absolute time for the termination of a task, e.g. June 23^{rd}, 2005; *interdependent temporal constraint*, for the definition of task start time and end time related to start/end of another task.

To the best of our knowledge, the problem of scheduling inside WfMSs has been considered in a limited way. Some techniques apply a first-in-first-out (FIFO) criterium, other ones present tasks to executors who choose the order

Proceedings of the Thirteenth International Symposium on Temporal Representation and Reasoning (TIME'06)
1530-1311/06 $20.00 © 2006 IEEE

of execution according to a random criterium [11]. In [11], the authors consider the problem of scheduling in WfMSs where there are uncertainties on the case processing times and routing. In [10], the authors focus specifically on resource allocation constraints in workflow scheduling: indeed, usually resource in workflows are limited, not shareable, and involve different execution costs.

3 Agent Temporal Modelling

The *organizational* model describes the structure of the organization where *cases* are enacted. The *scheduler* of the WfMS is entitled for task assignment: as a (predecessor) task is completed, the scheduler selects the (successor) task and retrieves from the process model the criteria for agent selection. The scheduler assigns a task (also named work item) to an agent, puts that task in that agent's work list, and forces that agent to accept the assigned task: the agent will execute the tasks on a "first-come, fist-served" basis.

A complete atemporal (timeless) description of an organization can be found in [8]. The core entity of the organizational model is *Agent*, which is the executing unit. Every task comes with a *Role* the agent selected for the execution must own. An *Actor* is a single executing unit, either human or machine, whose availability is under control: an actor can specialize, to identify particular responsibilities. A *Group* includes several agents which belong to the same project or to the same organizational unit: nested groups are permitted. A *Function* defines the different responsibilities/duties (i.e., what one can do and what one is allowed to do) inside the organization, and can be performed by a group or by one single actor. A *Team* describes a set of functions with regards to the organizational structure and may contain more agents with the same function.

Some temporal aspects must be considered to suitably model the organization, e.g. when the agent is present inside the organization or when an agent owns the suitable role. We extended the organizational model of [8] to deal with the *valid time* of the represented information, i.e. when the fact is true in the considered application domain [5]. To conceptually model the temporal requirements we adopt here a temporally extended ER model [5], where temporal entities (relationships), i.e. entities (relationships) with a valid time, are graphically depicted with a small watch: the overall organizational model is depicted in Figure 1.

The organizational model includes the temporal entity *Agent* and its temporal relationship with *Role*: other entities derived as specialization of *Agent* inherit temporal properties from *Agent* itself. Further temporalities are explicitly added by the entities *Availability* and *Unavailability*. *Availability* describes the normal availability of an agent, i.e. the normal working hours and working days of that agent. *Unavailability* describes when the agent is not available: this may happen either during the normal availability time for several reasons (e.g., illness, family reasons ...), or during local or nation-wide or world-wide holidays, or when the agent left under authorization (e.g., during an external mission). Both *Availability* and *Unavailability* represent a set of intervals (instants) where an agent can be available or unavailable. Valid times for these entities refer to the interval during which the given availabilities/unavailabilities hold. We want to stress the concept that the set of instants that belong to the *availability* is different from the set of instants that do not belong to *unavailability*: indeed, unavailability refers to the fact that an agent can be unavailable during his/her normal *availability* time, e.g. because of one of the above mentioned reasons. In the following we shall consider human agents, even though availability/unavailibility can be defined for automatic agents, too.

In the logical schema derived from the temporal organizational model of Figure 1, we shall focus on the tables corresponding to the entities *Agent*, *Role*, *Availability*, and *Unavailability* and their relationships. We consider the table `Agent` (Table 1) storing data about all the agents: the attribute `VT` describes the temporal interval of validity for the entire tuple, i.e., from when that agent was hired (lower bound) to when he/she left the organization (upper bound). If the upper bound of `VT` is $+\infty$, the agent still is on duty. The table `Role` (Table 1) describes all the roles available inside the organization. The table `Performance` (Table 1) describes the current (and past, if any) role of every agent, and it joins data from the tables `Agent` and `Role` (Table 1): as usual, `VT` describes the temporal interval of validity. As an example, from tables `Agent`, `Role`, and `Performance` (Table 1) it can be easily observed that *Charlotte Bronte* served as *Secretary* from 10-01-2001 to 31-08-2004 and she also has been serving as *Committee Member* since 01-09-2004 till now.

3.1 Temporal Information for the Organizational Model

To assign a task, the *scheduler* reads the process model and selects the agent mainly according to the required *role*. Several criteria can be used in task assignment policies: *cost* constraints consider economical aspects (e.g., the hourly cost of the agent) and also temporal aspects (e.g., to respect the deadline for execution, considering when the selected agent will complete the task); *control* constraints describe skills required to an agent, resource allocation policies (e.g., one unique executor for task B and task A). In this paper, we shall focus both on cost constraints, to identify deadlines for task execution as well as availability of agents, and on control constraints, to identify the role and the skills required to an agent to be selected for task execution.

Agent	Id_A	Name	Email	...	VT
	1	Jane Austen	jane.austen@literature.com	...	[10-01-2001÷+∞]
	2	James Joyce	james.joyce@literature.com	...	[10-01-2001÷+∞]
	3	Emily Bronte	emily.bronte@literature.com	...	[01-03-2003÷+∞]
	4	Charlotte Bronte	charlotte.bronte@literature.com	...	[10-01-2001÷+∞]
	5	Charles Dickens	charles.dickens@literature.com	...	[10-04-2000÷+∞]
	6	Samuel Beckett	samuel.beckett@literature.com	...	[10-01-2004÷+∞]

Role	Id_R	Description
	1	Secretary
	2	Committee Member
	3	Committee President

Performance	Actor	Role	VT
	1	1	[10-01-2002÷+∞]
	2	2	[10-01-2001÷+∞]
	3	1	[01-03-2002÷+∞]
	4	1	[10-01-2001÷31-08-2004]
	4	2	[01-09-2004÷+∞]
	5	3	[10-04-2000÷+∞]
	6	2	[10-01-2004÷+∞]

Table 1. The table `Agent` depicts data about agents; the table `Role` includes all the roles inside the organization; the table `Performance` describes the current and the past roles owned by every agent

Periodicity. When considering temporalities, the valid time does not suffice to cover all the required aspects. In fact, the availability/unavailability of an agent may present some periodicity.

The availability/unavailability of an agent can look like: every Monday 8:00a.m.÷4:30p.m.; the first Wednesday of every month 1:30p.m.÷5:30 p.m.; every day 8:30a.m.÷4:30p.m. In particular, in this work we restrict ourselves to denote the interval of hours during which an agent is available/unavailable on a given set of days, expressing them by a suitable notation, based on that proposed by Leban et al. [7]. The availability/unavailability is expressed by two types of expression. *Daily time* represents a temporal interval of hours for each day. *Periodic time* represents an interval of hours during the days specified by a periodic expression as: *Days_in_Weeks*, corresponding to the notation `Days:during:Weeks:during:Years`; *Days_in_Months*, corresponding to the notation `Days:during:Months:during:Years`; *Weeks_in_Months*, corresponding to the notation `Weeks:during:Months:during:Years`. We do not consider other semi-periodic constraints, e.g. 3 days a week, 2 times a day, 2 weeks a month. Table 2 depicts the periodic expressions in our temporal organizational model.

An Example. After having introduced some fundamental concepts about temporalities in managing agents inside an organization, some additional tables can complete our analysis. The table `Availability` (Table 3) describes the several intervals of availability that can be defined inside an organization. The attribute `Time_Av` may assume two different values: i) "`DAILY_TIME`" indicates that the information is related to an interval repeating every day; ii) "`PERIODIC_TIME`" indicates that the information is related to a periodic time, which will be represented by the periodic expressions, depicted in Table 2. The attributes

Expression	P	Example
`[P]/Days_in_Weeks`	x such that x ∈ [1,..,7]	`[1]/Days_in_Weeks` = every Monday
	1..x such that x ∈ [2,..,7]	`[1..3]/Days_in_Weeks` = the first 3 days of the week
	x,y such that x,y ∈ [1,..,7]	`[1,3]/Days_in_Weeks` = every Monday and every Wednesday
`[P]/Days_in_Months`	1..p such that p ∈ [2,..,M] M ∈ [28,..,31]	`[1..5]/Days_in_Months` = the first 5 days of the month
	p,t such that p,t ∈ [1,..,M] M ∈ [28,..,31]	`[2, 6]/Days_in_Months` = day 2 and day 6 of the month
	w̄ such that w̄ ∈ [1,..,7]	`[1]/Days_in_Months` = the first Monday of the month
	p such that p ∈ [1,..,M] M ∈ [28,..,31]	`[20]/Days_in_Months` = day 20 of the month
`[P]/Weeks_in_Months`	1..q such that x ∈ [2,..,5]	`[1..2]/Weeks_in_Months` = the first 2 weeks of the month
	q such that q ∈ [1,..,5]	`[1]/Weeks_in_Months` = the first week of the month
	q,r such that q,r ∈ [1,..,5]	`[2,4]/Weeks_in_Months` = the second and the fourth week of the month

Table 2. The supported periodic expressions: notations and examples

`Start_Time` and `End_Time` relate to every single interval of availability: e.g., the first row, whose valid time is from 01-06-2005 to 30-11-2005, indicates that the working hours of all the agents sharing that tuple (i.e., agents that share that interval of availability) start at 9:00:00 and complete at 17:30:00, for the days inside the valid time of the tuple. Join paths from the table `Agent` to the tables `Availability` and `Unav` are performed via the tables `Accessibility` and `Unaccessibility`.

The fourth tuple of the table `Availability` (Table 3) is labelled as "`PERIODIC_TIME`": thus, the availability has some periodicity that is clarified by the corresponding tuple (`Id` = 4 and `Type` = "`Av`") in the table `Periodic_time`,

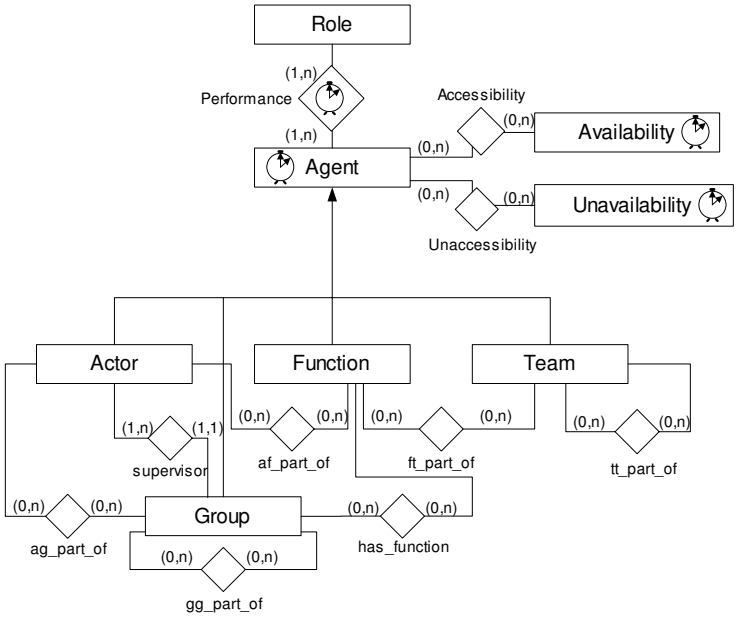

Figure 1. A temporal organizational model

whose `Expression` is `[1]/Days_in_Weeks`: the periodicity considers every Monday (first day) of every week. Analogously, the table `Unav` describes the several intervals of unavailability that can be defined inside an organization. The attribute `Time_Unav` may assume the values "`DAILY_TIME`" and "`PERIODIC_TIME`", with the same meaning of the corresponding values of the attribute `Time_Av` for table `Availability`. The attribute `Description` briefly describes the reason of unavailability, while the attributes `Start_Time` and `End_Time` relate to every single interval of unavailability: e.g., the first row, whose valid time is from 01-09-2005 to 01-09-2005, indicates that all the agents sharing that tuple (i.e., agents that share that interval of unavailability) are not working in the span of hours starting at 9:00:00 and completing at 17:30:00, for the day identified by the valid time of the tuple. Since the type of the tuple is "`DAILY`", the periodicity is defined on a daily basis and no related tuple can be found in the table `Periodic_Time`.

3.2 Temporal Requirements from the Process Model

Beside considering the temporalities of agents, the scheduler needs also additional information from the process model to include for every task two relevant temporal attributes [2]: `expectedTaskDuration` defines a reasonable amount of time required by an agent to complete the task; `maxTaskDuration` defines the maximum amount of time the process can accept for the agent to complete the task. Briefly speaking, `expectedTaskDuration` is used by the scheduler to foresee the real amount of time needed to complete the task, and thus to balance the workload among agents; `maxTaskDuration` is used by the scheduler to fix the deadline for the completion of the task. The process model can also be enriched by many other constraints: for instance, a constraint may state that the successor task must be opened within a given temporal delay after the predecessor task was completed. For this constraint, the expected instant of opening the task by the agent is related to the real amount of time required by the agent to complete the tasks already included into his/her work list.

As an example, consider a simple schema made by a straight connection from task A to task B. A is completed at time t_{endA}. The constraint in the process model requires that the successor task B must be started within a temporal delay δ after the completion of A. Regardless of the organizational model, the constraint can also be expressed as: t_{startB} - $t_{endA} \leq \delta$. If the process model requires for task B an agent with role *Secretary*, and his/her work list already includes task C (i.e., the task comes from a different process and it has not been started yet, but it will start at t_{startC} and finish at t_{endC}), the agent is selectable for the execution of B if $(t_{endC} - t_{endA}) < \delta$ (see Figure 2).

If the property $(t_{endC} - t_{endA}) < \delta$ does not hold, a different agent must be selected. Should no agent with the suitable role (*Secretary*) be available to execute task B, the task assignment is hierarchically escalated till the nearest agent, whose role complies with the specified role, is found.

Accessibility	Agent	Availability
	1	1
	2	2
	3	3
	4	5
	5	4
	6	6

Inaccessibility	Agent	Unavailability
	1	1
	2	4
	3	3
	4	3
	5	2
	6	3

Availability	Id_Av	Start_Time	End_Time	Time_Av	VT
	1	09:00:00	12:00:00	DAILY_TIME	[01-06-2005÷30-11-2005]
	2	10:00:00	13:30:00	DAILY_TIME	[01-06-2005÷30-11-2005]
	3	09:00:00	14:00:00	DAILY_TIME	[01-06-2005÷30-11-2005]
	4	10:00:00	12:30:00	PERIODIC_TIME	[01-07-2005÷31-10-2005]
	5	13:00:00	17:30:00	DAILY_TIME	[01-06-2005÷30-11-2005]
	6	09:00:00	17:30:00	DAILY_TIME	[01-06-2005÷30-11-2005]

Unav	Id_Unav	Description	Start_Time	End_Time	Time_Unav	VT
	1	Holiday	09:00:00	17:30:00	DAILY_TIME	[01-09-2005÷01-09-2005]
	2	BusinessTrip	09:00:00	17:30:00	PERIODIC_TIME	[01-09-2005÷30-11-2005]
	3	Holiday	09:00:00	17:30:00	DAILY_TIME	[10-08-2005÷20-08-2005]
	4	BusinessTrip	09:00:00	17:30:00	PERIODIC_TIME	[01-09-2005÷30-11-2005]

Periodic_Time	Id	Type	Expression
	4	Av	[1]/Days_in_Weeks
	2	Unav	[2]/Weeks_in_Months
	4	Unav	[1..5]/Days_in_Months

Table 3. `Accessibility` **and** `Inaccessibility` **depict data about the** *availability* **and the** *unavailability* **of every single** *agent***.** `Availability` **depicts data about the** *availability* **intervals of all the** *agent***s.** `Unav` **depicts data about the** *unavailability* **intervals of all the** *agent***s.** `Periodic_Time` **clarifies the type of availability or of unavailability indicated by the attributes** `Time_Av` **and** `Time_Unav` **of tables** `Availability` **and** `Unav`**, respectively**

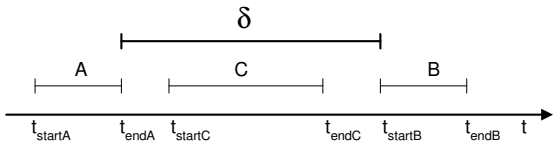

Figure 2. Selection of agent *Secretary* **for the execution of task** B**: the assigned task** C **must finish within the specified delay.**

4 The Temporal Scheduler

Policies of a *temporal* scheduler must consider the real *availability* of the agent, as well as the *unavailability* of the agent, and the *presence of the agent at task start time*. As an example, consider an agent on holiday till the instant $t_{AgentBackToWork}$: any task assigned to that agent will start to be executed only after the selected agent will have come back, i.e. after $t_{AgentBackToWork}$. In this situation, the expected deadline for completion of the task ($t_{Deadline}$) cannot be guaranteed, if the relationship among the instants $t_{Deadline} < t_{AgentBackToWork}$ holds. As a second example, consider the constraint of the process model stating that the successor task must be started within a maximum delay from the completion of the predecessor task: call $t_{maxStartDelay}$ this deadline for the start of the execution. If the selected agent will be available only after the instant $t_{maxStartDelay}$, i.e. the relationship $t_{maxStartDelay} < t_{AgentBackToWork}$ holds, that agent cannot be selected as executing agent, or the constraint will be surely violated.

The temporal scheduler has to consider several aspects, as the expected start time for task execution (i.e. when the selected agent will effectively start the execution), and also the `maxTaskDuration`, to check whether the availability of the agent may suffice for the completion of the task: if no agent is found, the scheduler considers the `expectedTaskDuration`, to estimate whether the considered agent could be able to complete the task within this second duration. In this latter case, we cannot say for sure if the agent will complete the task even in its worst case.

If no agent is selectable according to the currently owned role, an alternate agent must be found. The task assignment policy is hierarchically escalated to find the agent whose past role is nearest to the currently required role and whose availability complies with the temporal requirements. In this search, the temporal properties suitably managed by a Temporal DBMS are fundamental.

4.1 The Algorithm

The algorithm for task assignment considers the tasks to be assigned and sorts them according to their priority. The priority is evaluated according to the expected deadline for completion of the task (a task whose deadline is the nearest comes first) and, secondly, according to the expected duration of the task (a task whose duration is the shortest comes first): in this way, tasks whose deadline for completion is the nearest will be scheduled first.

The task assignment policy considers the selection criteria in the following order: role, effective availability, workload, number of unavailabilities, presence at task start and task completion times. *Role*: the selectable agents must own the role defined by the process model. *Effective availability*: the selectable agents are those who can complete the task within the specified deadlines, with respect to the effective working time of the agents. *Workload*: the workload of task executions must be balanced among the agents; agents whose recent history presents a lower workload will be selected first. *Number of unavailabilities*: agents with a lower number of unavailabilities over the expected interval of execution of the task will be selected first. *Presence at task start and task completion times*: the selectable agents are those who can guarantee their availability during the expected execution time for the task.

The algorithm consists of several functions. The main function `ATWC` - *A*ssign *T*ask *W*ith *C*onstraints - assigns a task to an agent according to the defined constraints. `ATWC` sorts the tasks to be assigned according to their priority: Figure 3 depicts the pseudo code for the function `ATWC`. The Appendix briefly discusses the complexity of `ATWC`.

The first check on agents is performed by the function `verifyR`, verifying the *role* of the selectable agent. The second check on these agents is performed by the function `verifyTAV`, aiming at verifying the temporal availability of the agents. This latter function `verifyTAV` aims at verifying the real availability of the agent and considers all the involved temporalities. Actually, `verifyTAV` checks whether the selectable agent is available for a span of time that guarantees the completion of the task within the defined constraints: the span of time obtained by considering both the *availability* and the *unavailability* of that agent must be greater than the `maxTaskDuration` of the task under consideration plus the amount of time required by the

```
INPUT      List R of resources
           List T of tasks to be assigned, sorted by descending priority
OUTPUT     bAssignSolved, boolean var: TRUE if assignment is successful
METHOD     boolean bTryUsingPastRole = FALSE;
           boolean bAssign = FALSE;
           bAssignSolved = FALSE;
           list resourceR;
           resource resourceAV;
           for each t ∈ T do
             bTryUsingPastRole = FALSE;
             resourceR = verifyR(t, R, bTryUsingPastRole);
             resourceAV = verifyTAV(t, resourceR);
             if (resourceAV = null) then
                bTryUsingPastRole = TRUE;
                resourceR = verifyR(t, R, bTryUsingPastRole);
                resourceAV = verifyTAV(t, resourceR);
             endif
             if (resourceAV ≠ null ) then
                bAssign = doA(t, resourceAV, bTryUsingPastRole);
                T = T \{t};
             endif
           end for
           if (T = ∅ ) then
                bAssignSolved = TRUE;
           return bAssignSolved
```

Figure 3. Pseudo code for the algorithm `ATWC` - *A*ssign *T*ask *W*ith *C*onstraint.

agent to execute all the tasks that already are inside the work list of that agent. In other words, `verifyTAV` has to check whether the following property holds for the agent j who is the candidate agent for the execution of task k and whose work list includes all the i tasks:

$$effAvailability_j > maxTaskDuration_k + \sum_{i=1}^{n} maxTaskDuration_i \quad (1)$$

where $effAvailability_j$ (i.e., the effective availability) is a span of time (i.e., a duration) obtained by considering the intervals of availability and unavailability of the agent j, within the time constraints for the completion of the task.

If two or more agents are selectable, the function `verifyTAV` compares the workload of the selectable agents. Let us assume that the agent j has the work list WL_j, including all the tasks i; the workload $\mathcal{W}_j$ of the agent j can be defined as:

$$\mathcal{W}_j = \sum_{i \in WL_j} maxTaskDuration_i \quad (2)$$

The scheduler will select the agent having the minimum $\mathcal{W}_j$, among all the selectable agents. If two or more agents still share the same workload, the function `verifyTAV` compares these agents and selects the agent whose available working time is greater if considered over the lifespan of the task. The available working time is evaluated by considering the *availability* and the *unavailability* over the lifespan of the task. The last check is on the presence of the agents at the expected start and completion times of the

tasks; eventually, if two or more agents still persist at the same level, the head or tail mechanism is invoked. If, instead, the `verifyTAV` function returns no agent available, the algorithm tries, with an approach similar to that just explained, to extract an agent whose *past* role is compliant with the role required for the task under assignment. At the end, if after the run of `verifyTAV`, one agent remains as selectable, the `doA` function completes the assignment by putting the task k inside the work list of the selected agent j.

A critical step of this algorithm deals with periodicities of the *availability* and *unavailability* of an agent. Periodicities are suitably managed by the functions `extractAv` and `extractUnav`, which extract the *availability* and *unavailability* for a given agent. These functions consider the periodicities, expressed as *Day_in_Weeks*, *Days_in_Months*, and *Weeks_in_Months* (Section 3.1), and compute their occurrence over the expected lifespan of the task.

4.2 A Detailed Example

To validate the above methodology, we implemented a prototype. The process considered is that of the PhD candidate enrollment. An applicant sends in an application form for a PhD candidate position; applicants' CVs are analyzed first and applicants whose CV is passed will be interviewed, only; finally, according to the result of the interview, candidates will be enrolled into the position. The database storing the process model, the organizational model, and the information model has been implemented as described in [3].

As an example, we consider the organizational model of Tables 1÷3, and we have to select agents for the task `Interview` of case with `Id`=27. ¿From the process model, the task requires an agent with the role of `Committee Member`. From the tables `Role` and `Performance`, we can identify that the selectable agents have `Id_A` 2, 6, and 4 - since the current timestamp is after September 1^{st}, 2004-.

We assume the current timestamp is Saturday October 1^{st}, 2005, and the deadline for the completion of task `Interview` for case with `Id`=27 is Wednesday October 5^{th}, 2005. Considering the tables `Accessibility` and `Inaccessibility` of Table 3, the tuples of the table `Availability` (Table 3), related to the agents whose current role fits with the required role, are those with `Id_Av`=2 (for the agent with `Id_A`=2), `Id_Av`=5 (for `Id_A`=4), `Id_Av`=6 (for `Id_A`=6). The tuples of the table `Unav` (Table 3), related to the agents whose current role fits with the role defined by the process model, are those with `Id_Unav`=3 (for the agents with `Id_A`=4 or `Id_A`=6), and `Id_Unav`=4 (for `Id_A`=2). We also assume that the work lists of the agents 2, 4 and 6 are empty.

As there is at least one tuple selected from the tables `Availability` and `Unav` that may come with some complex periodicity (where `Time_Unav`="`PERIODIC_TIME`"), we also have to consider the tuples from the table `Periodic_Time` where `Type`="`Unav`". The selected tuple from the table `Periodic_Time` has `Expression`="`[1..5]/Days_in_Months`" (the periodic unavailability concerns the first 5 working days of every month): this unavailability is related to the agent with `Id_A`=2, only. As it can be easily observed, the agent with `Id_A`=2 is not selectable for the task `Interview` for case with `Id`=27 as she, even though usually available every day from 10 a.m.÷12:30 p.m. (Table 3), is not available in the first 5 days of October and will only be available after the deadline of the task. The selectable agents are thus characterized by `Id_A`=4 or `Id_A`=6. From the table `Availability` (Table 3), the tuple related to agent 6, i.e. the tuple with `Id_Av`=6, is characterized by a wider working time. Therefore, the selected agent will be agent 6.

5 Discussion and Conclusions

In this paper we proposed some extensions to the (atemporal) workflow organization model and to the task assignment policies adopted by the scheduler of a WfMS. Typical policies defined for the scheduling process are: the agent must own a suitable role; the workload of agents with a same role must be balanced. The proposed extensions aim at managing temporal aspects when selecting an agent for the execution of a task, and concern both the constraints defined in the process model and in the organizational model. More specifically, the constraints defined in the organizational model may require to consider the *availability* and *unavailability* times of an agent, which can be expressed - and very often are - as periodic temporalities: e.g., an agent is available every first Monday of a month for the full day, from Tuesday to Saturday for the afternoon, only.

As a proof-of-concept, a running prototype implements the algorithms of the temporal scheduler for a WfMS. The software system has been developed by the Java programming language, on top of an Oracle DBMS with a temporal layer provided by TimeDB [3].

Acknowledgements. We are grateful to Sara Allegretti, and to Stefano Frigerio and Clara Sonzogni for developing and implementing the running prototype.

References

[1] C. Combi and G. Pozzi. Towards temporal information in workflow systems. In S. Spaccapietra, S. T. March, and Y. Kambayashi, editors, *ER (Workshops)*, volume 2503 of *LNCS*, pages 13–25. Springer, 2002.

[2] C. Combi and G. Pozzi. Temporal conceptual modelling of workflows. In I.-Y. Song, S. W. Liddle, T. W. Ling, and P. Scheuermann, editors, *ER*, volume 2813 of *LNCS*, pages 59–76. Springer, 2003.

[3] C. Combi and G. Pozzi. Architectures for a temporal workflow management system. In Haddad et al. [6], pages 659–666.

[4] J. Eder, W. Gruber, and E. Panagos. Temporal modeling of workflows with conditional execution paths. In M. T. Ibrahim, J. Küng, and N. Revell, editors, *DEXA*, volume 1873 of *LNCS*, pages 243–253. Springer, 2000.

[5] H. Gregersen and C. S. Jensen. Temporal entity-relationship models - a survey. *IEEE Trans. Knowl. Data Eng.*, 11(3):464–497, 1999.

[6] H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors. *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004*. ACM, 2004.

[7] B. Leban, D. McDonald, and D. Forster. A representation for collections of temporal intervals. In *AAAI86*, pages 367–371, 1986.

[8] Paul Grefen and Barbara Pernici and Gabriel Sánchez (eds). *Database Support for Workflow Management*. Kluwer Academic, Dordrecht, 1999.

[9] W. Sadiq, O. Marjanovic, and M. E. Orlowska. Managing change and time in dynamic workflow processes. *Int. J. Cooperative Inf. Syst.*, 9(1-2):93–116, 2000.

[10] P. Senkul, M. Kifer, and I. H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *VLDB*, pages 694–705, 2002.

[11] G. B. Tramontina, J. Wainer, and C. A. Ellis. Applying scheduling techniques to minimize the number of late jobs in workflow systems. In Haddad et al. [6], pages 1396–1403.

Appendix A: Complexity Issues

Let A be the set of agents, T be the set of tasks to be assigned, R be the set of roles available in the organization, and a, t, r be integer positive numbers such that $|A| = a$, $|T| = t$, $|R| = r$. The algorithm complexity of the function `verifyR` is $O(a \times r)$: the function considers all the resources and all the roles of the organization.

The algorithm complexity of the function `verifyTAV` relates to the complexity of the algorithms of the invoked functions `extractAv` and `extractUnav`, which extract for a given resource the availability (unavailability) intervals. These latter functions may deal with periodic expressions, considering the 365 days of the year (the worst case), or the 52 weeks of the year, or the 12 months of the year. If we assume as the reasonable working period (from the enrollment to the retire) for a person an amount of years less than 100, the complexity introduced by the periodic calendar can be estimated as a constant. Then, for each periodic expression, we have to evaluate the intervals of availability (unavailability): if we limit our hypothesis to a granularity of seconds, for a period of 100 years we have a grand total amount of seconds as $100 \times 365 \times 12 \times 24 \times 60 \times 60$, which is another constant value. Thus, we can consider the overall complexity of the algorithms through the constants C_{av} and C_{unav} for the functions `extractAv` and `extractUnav`, respectively. Moreover, the function `verifyTAV` has to invoke some other functions for each considered agent: (i) `checkEffectiveAv` computes the effective availability of each agent (considering availabilities, unavailabilities, and assigned tasks); (ii) `checkLWR` (check lower workload resource) looks for the agent with the lowest value of workload; (iii) `checkLUnav` looks for the agent with the lowest value of unavailability; and (iv) `checkPresence` considers all the availabilities and all the unavailabilities at task start time and at the expected task completion time. The algorithm of the function `checkEffectiveAv` is not related to a, t, r, and thus its complexity can be expressed through a constant C_{effAv}. The algorithm of the function `checkLWR` may have to consider, as the worst case, the agents twice: thus its order of complexity is $O(2a)$. The algorithm of the function `checkLUnav` has to consider the agents only once: thus its order of complexity is $O(a)$. Finally, the algorithm of the function `checkPresence` has to scan once the availabilities and twice the unavailabilities: thus the order of complexity for the algorithm is $O(C_{av}+2C_{unav})$. By putting all of these complexities together, we obtain for the algorithm of `verifyTAV` the following complexity $\mathcal{O}$:

$$\begin{aligned}\mathcal{O} \quad = \quad & O(a \times (C_{av} + C_{unav} + C_{effAv} + 2a + a + \\ & C_{av} + 2C_{unav})) \end{aligned} \tag{3}$$

The grand complexity of the algorithm of the function `verifyTAV` is $O(a^2)$. The complexity of the algorithm of the function `doA`, which assigns the task to the selected agent, is constant: let us call it C_d. Finally, the complexity $\mathcal{C}$ of the algorithm `ATWC`, related to that of `verifyR`, `verifyTAV`, and `doA`, is:

$$\mathcal{C} \quad = \quad O(t \times (a \times r + a^2 + C_d)) \tag{4}$$

which is $O(t \times a \times r + t \times a^2)$. If l is the maximum value of t, a, r, the complexity is $O(l^3)$. The grand complexity of the entire algorithm is thus *polynomial*.

Performance Evaluation. In a real organization adopting a WfMS, managed processes generally are quite complex, include a great number of tasks, and thus a high value of t. The number a of available agents is reasonably high. Agents, then, generally belong to a restricted number of possible roles r.

If we consider the formula $O(t \times a \times r + t \times a^2)$, and from the above we assume the hypothesis that $r \ll t$, and $r \ll a$, then we can approximate the order of complexity to $O(t \times a^2)$. Finally, we also want to outline that the finer the granularity becomes, the heavier the parameters C_{av} and C_{unav} are.