Mixed-Initiative Management of Online Calendars

L. Ardissono, G. Petrone, M. Segnan, and G. Torta

Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, Torino, Italy
{liliana.ardissono,giovanna.petrone,marino.segnan,gianluca.torta}@unito.it

Abstract. Calendar management has been recognized as a complex, highly personal type of activity, which must take individual preferences and constraints into account in the formulation of satisfactory schedules. Current calendar management services are affected by two limitations: most of them lack any reasoning capabilities and thus cannot help the user in the management of tight schedules, which make the allocation of new tasks particularly challenging. Others are too impositive because they proactively schedule events without involving the user in the decision process.

In order to address such issues, we propose a mixed-initiative approach which enables the user to select the events to be considered, receive safe schedule suggestions from the system and select the preferred ones for revising a calendar. A peculiarity of our system is the fact that, in the suggestion of alternative schedules for an event, it searches for solutions which are very similar to the user's current schedule, with the aim of limiting changes to her/his daily plans as much as possible. Our calendar management service is based on the exploitation of well-known Temporal Constraint Satisfaction Problems techniques, which guarantee the generation of safe scheduling solutions.

Keywords: Calendar scheduling, mixed-initiative interaction, temporal reasoning.

1 Introduction

The recent adoption of Web-based calendars has empowered people to holistically handle their own life schedules by exploiting ubiquitous services for organizing their work and personal commitments; e.g., see [1] for a discussion on the importance of this issue. However, current calendar services are affected by limitations which reduce their usefulness: on the one hand, many services have no scheduling capabilities and thus leave the user alone in the resolution of timing conflicts, which may occur in tight schedules and are particularly difficult to handle, especially if they involve multiple actors. On the other hand, as discussed in [2], the fully or semiautomated scheduling systems developed so far "fail to account for the personal nature of scheduling, or they demand too much control of an important aspect of an individual's working world." A new, mixed-initiative scheduling model is thus needed to mediate between too little and too much proactivity: the idea is that of enabling the user to steer and control the system's operations, in order to receive a scheduling support that meets individual needs and preferences.

As a first step towards addressing this issue, we propose an intelligent, mixed-initiative scheduler supporting the management and revision of the user's calendar, given her/his commitments and those of the other actors involved in the shared activities. The main feature of our scheduler is its *mixed-initiative, conservative support*: besides the generation of complete schedule proposals, it helps the user to revise a schedule by suggesting alternative allocations of the events/tasks to be moved, having schedule stability as a priority.

- The mixed-initiative support is implemented as follows: by interacting with the rich user interface offered by our system, the user can select the events to be moved and explore their alternative allocations in order to decide which one should be applied.
- As far as stability is concerned, in the generation of the revision hypotheses, the system proposes conservative changes to the user's calendar in order to maintain previous commitments as originally planned or with minor temporal shifts.

Our scheduler, an initial version of which was presented in [3], is based on the exploitation of Temporal Constraint Satisfaction Problems (TCSP) techniques, used to generate safe full schedule proposals as well as to present all of the admissible intervals for the placement of specific tasks. At present, the system only offers a user interface for standard Web browsers, but we will soon develop an analogous one for mobile devices.

In the remainder of this paper, Section 2 provides some background on event scheduling and compares our work to the related one. Then, Section 3 describes a usage scenario and the features offered by our mixed-initiative scheduler. Section 4 describes the system and its underlying model. Section 5 provides some technical details and section 6 concludes the paper.

2 Background

The temporal allocation of events in calendars has been addressed from different viewpoints, offering more or less complex features for event scheduling, but the research on mixed-initiative interaction with the user has been rather limited so far. In the following, we briefly review the main types of support offered by commercial and research tools.

The simplest services are the to-do-list managers, such as Remember The Milk [4], which are connected to the user's calendar but typically do not provide any scheduling support. They only present the lists of items allocated in each time slot.

Most calendar management services (e.g., Google Calendar [5]) do not offer any scheduling function: they only help the user to identify free time slots for shared tasks and meetings by jointly visualizing the calendars of the actors to be involved, or by listing the free time slots to choose from (e.g., see the Google Calendar smart schedule feature). Other tools take actors and resources into account (e.g., Resource Central [6] supports the allocation of meeting rooms, etc.) but have no scheduling capabilities either.

Similarly, task managers such as Things [7], DoIt [8] and Standss Smart Schedules for Outlook [9] support the management of tasks, deadlines and task dependencies but they do not schedule any tasks.

Temporal reasoning and scheduling have been introduced in some process management tools to address their lack of capability to reason about time. In [10] the authors make use of the *Oz* multi-paradigm programming language [11] for solving scheduling problems with CLP techniques similar to the ones used in our work. Some process management tools, such as the one described in [12], offer a complementary feature with respect of our work: they estimate the dates of execution of future tasks to help the user preview the organization of pending commitments. Other tools, such as WorkWeb System [13], schedule multiple workflows by taking the availability of actors and resources (e.g., meeting rooms) into account. The main role of the actors' personal schedulers is that of automatically (or manually) accepting or rejecting new tasks and modifications proposed by other agents.

Complex schedulers plan the execution of tasks according to deadlines and to the surrounding context, e.g., in mission planning and/or robotic applications; for instance, see Pisces [14]. However, they are not suitable for managing the user's daily schedules, either because they are developed for very specific execution environments, or because they focus on allocating physical resources, without taking people's needs into account.

Opportunistic schedulers, typically based on planning technology, synchronously guide the user in the execution of activities according to the pending goals to be achieved. However, they do not provide the user with an overview of long-term schedules, do not manage the shared activities and are not mixed-initiative: they basically suggest opportunities of action, which the user may accept or ignore. For instance, see [15].

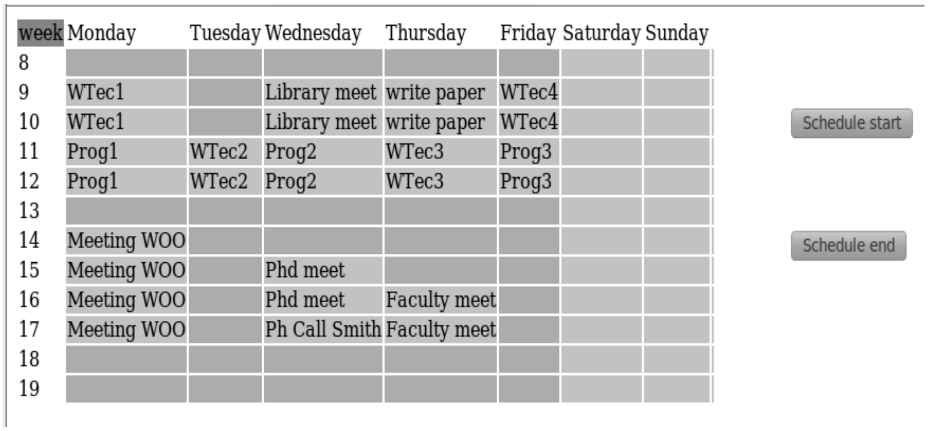
To the best of our knowledge, the only calendar management service which supports mixed-initiative scheduling is PTIME [2]. That system helps the user to organize personal and shared events by selecting high-level schedule generation criteria (e.g., favoring the robustness of the schedules, in terms of fault tolerance, or their tightness, and so forth). Our work differs in two ways with respect to PTIME:

– First, our system separates the selection of the scheduling policies to be applied from the identification of the portions of a calendar to be affected. It enables the user to revise a calendar by explicitly selecting the events and tasks to be rescheduled, and to preview the possible solutions (if any) to choose from.
– Second, our system supports the management of conservative schedule revisions which do not alter the relative order of the allocated tasks and events (except for the one selected for modifications). It proposes changes which involve sliding the other allocated items (e.g., postponing or anticipating them with respect to their current timing) and, as such, reduces the changes in the actors' calendars.

3 Sample Scenario

Let's consider the sample calendar in Figure 1, where a set of events has been allocated by user *U* using our mixed-initiative scheduler. As described later on, the user can (i) manually set events/tasks in the calendar; (ii) create events/tasks to be automatically allocated by the scheduler; (iii) select items to be moved to a different date and time (benefiting from the help of the system in such an activity).

The placement of tasks in the displayed calendar satisfies some constraints given by the user: for example, the Library meeting cannot take place at lunch time (13.00 to



week	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
8							
9	WTec1		Library meet	write paper	WTec4		
10	WTec1		Library meet	write paper	WTec4		
11	Prog1	WTec2	Prog2	WTec3	Prog3		
12	Prog1	WTec2	Prog2	WTec3	Prog3		
13							
14	Meeting WOO						
15	Meeting WOO		Phd meet				
16	Meeting WOO		Phd meet	Faculty meet			
17	Meeting WOO		Ph Call Smith	Faculty meet			
18							
19							

Fig. 1. User interface of our mixed-initiative scheduler: week calendar view

14.00; i.e., from 1pm to 2pm), nor after 17.00, and must end before Thursday 11.00. Moreover, the Ph.D meeting and the phone call must take place on Wednesday before 20.00 and the phone call must be done after the Ph.D meeting.

Let's assume that a new task arrives (e.g., meeting the plumber for fixing a leaking sink), that takes 3 hours and has to be performed on Wednesday before 18.00. It is easy to see that there is no free spot in the calendar where the new task can be placed. Then, the user can ask for the help of our scheduler: first, (s)he asks when the task can be allocated; in this case, the scheduler replies that it could start at 13.00, 14.00 or 15.00, since this would only require to anticipate or postpone the afternoon tasks, without affecting the *order* of the current ones. Indeed, if the new task is placed at 13.00 or 14.00, it is sufficient to delay a bit the Ph.D meeting and the phone call. Otherwise, if it is placed at 15.00, a solution can be found by anticipating the Ph.D meeting and deferring the phone call, and slipping the new task between them. If the user does not want to meet the plumber in the afternoon, (s)he can try to move the Library meeting to make room for the new task. For this purpose, the user points to the Library meeting task and asks the temporal reasoner for help. The reasoner replies that, considering only the user's tasks in the current schedule, the Library meeting can be moved to Wednesday at 8.00, 9.00, 14.00, 15.00 or to Thursday at 8.00 or at 9.00 (in which case, the write paper task should be anticipated to Wednesday afternoon).

Notice that if, after exploring several possibilities with the help of the scheduler, the user is still unsatisfied, (s)he can request a brand new schedule. However, this may cause many of the other tasks to change their times and their relative order.

4 Our Mixed-Initiative Scheduler

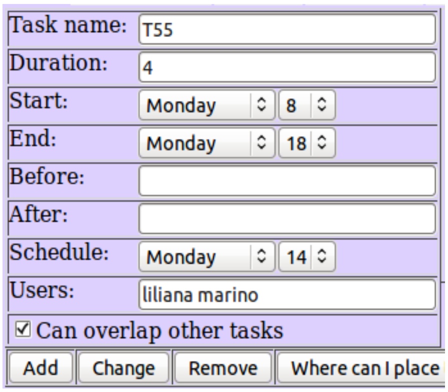
Our scheduler is integrated in a Collaborative Task Manager (CTM) service [16,17] that supports distributed collaboration by enabling users to synchronize with each other in the execution of their shared activities. The CTM manages task nets that regulate the execution of complex activities, possibly decomposed in simpler tasks which can be organized in patterns typical of workflow nets; e.g., sequence, parallel split, exclusive choice, synchronization, simple merge, etc. [18].

Our system manages calendar events and tasks which can involve multiple actors. In our view, the task concept subsumes the event one, as an event can represent a very simple "to-do", or an appointment not necessarily associated to the execution of specific operations. Thus, in the following we will only refer to tasks, assuming that events can be treated in the same way. The design of our scheduler has been driven by the following requirements:

- *Safe scheduling:* the proposed solutions must be consistent with the constraints imposed on the tasks in the user's calendar and with the commitments of the other involved actors; i.e., the scheduler must propose task allocations that are feasible for all the participants.
- *User control:* if the user wants to inspect the space of possible solutions, e.g., to allocate a new task, or to move an existing one, (s)he must be enabled to steer the system's behavior in order to select the paths to be explored. This is very important to impose personal scheduling preferences (e.g., by explicitly selecting the "victims" to be revised in order to solve a conflict).
- *Mixed-initiative:* even though the system has an important role in suggesting possible solutions to the existing conflicts, the user is in charge of exploring the available alternatives and selecting the preferred one. In other words, the user has an active role in guiding the scheduler's operations rather than being only responsible for accepting or rejecting the proposed solutions.
- *Conservativeness:* unless the user requests a new schedule, the system must search for solutions that are as conservative as possible with respect to the existing commitments in order to avoid a complete reorganization of the actors' daily schedules.
- *Collaboration support:* tasks are scheduled for all the involved actors, taking into account their calendars and the deadlines of their commitments.

Our mixed-initiative scheduler enables the user to define the tasks to be performed by specifying their actors, earliest start time, duration, deadline, and other types of information, such as, e.g., whether a task can be performed in parallel with other ones (i.e., it can overlap with other tasks). Even though some tasks have a fixed starting time, other ones can be scheduled at different time points and there is a safe starting time window which spans from the instant when they are enabled (earliest starting time) until the very last minute they have to be started to meet their deadline. In order to safely schedule a task, it has to be allocated within its safe starting time window. However, the specific allocation is not by itself a hard constraint to be met and can be modified for re-scheduling purposes. We thus model two main types of information: the basic constraints of tasks, which have to be met in any schedule proposal, and the specific configuration of a calendar, which represents the user's current decisions about how to organize the activities, but can be modified. The representation of tasks, and the temporal reasoning approach adopted in our work, reflect this idea.

As discussed later on, a critical aspect concerns the execution of shared tasks, whose scheduling affects multiple actors. Our current system fully addresses the management of a single calendar but provides a partial solution to the synchronization of multiple calendars, to be further developed in our future work.



Task name:	T55
Duration:	4
Start:	Monday 8
End:	Monday 18
Before:	
After:	
Schedule:	Monday 14
Users:	liliana marino

☑ Can overlap other tasks

Fig. 2. User interface of our mixed-initiative scheduler: event/task specification

4.1 User Interface

Figure 1 shows a portion of the user interface of our mixed-initiative scheduler. This user interface, currently only available for standard web browsers, is aimed at testing the scheduling capabilities of our prototype. We will restyle it, and develop another one supporting mobile access (mainly for tablets, as schedule revision is a rather difficult task to be performed using a smart phone), after having collected feedback from our users.

– An interactive calendar shows the user's schedule for the current week by displaying the names of the tasks in the time slots that have been associated with them.
– By clicking on a cell of the calendar the user can view and modify the details of the allocated task or delete it. Figure 2 shows the portion of the user interface offering this functionality (see the "Change" and "Remove" buttons).
– Figure 2 also shows the portion of the user interface supporting the definition of new tasks ("Add" button) and the rescheduling of tasks ("Where can I place this task?" button, partially displayed in the figure). If the user clicks on the "Where can I place this task?" button, the system visualizes in a pop-up window the safe task allocations that could be selected to revise the overall schedule, possibly by sliding other tasks in order to make room for the selected one. All such alternatives are handled as revision hypotheses and it is up to the user to decide whether applying one of them (thus updating the calendar) or not.
– In Figure 1, at the right of the calendar, the *Schedule start* and *Schedule end* buttons enable the user to request a new schedule following different task allocation policies. If at least one schedule solution exists, the *Schedule start* policy proposes one where tasks that can be started earlier are allocated before the others. Differently, the *Schedule end* policy produces a schedule where tasks are allocated depending on their urgency, i.e., those having earlier deadline are allocated before the others. The former policy produces tighter schedules, reducing the free time slots in the user's calendar. The latter is more cautious and generates more robust schedules by reserving time after the expected termination of tasks, which might be possibly exploited for recovery purposes.

4.2 Scheduling Modules

The mixed-initiative scheduling service offered by our system is based on the integration of two main software components:

- A basic scheduler (henceforth, scheduling module), which supports the generation of brand new schedules satisfying the temporal constraints of the existing tasks.
- A temporal reasoner suggesting alternative allocations for a specific task in the current calendar.

The scheduling module, given a set of tasks, their definition (e.g., duration, earliest start time and deadline) and the allocation policy selected by the user attempts to place the tasks in the calendars of the involved actors and proposes a solution, if any. Unless specified by the user by checking the "Can overlap other tasks" box in the task definition form (see Figure 2), we assume that tasks cannot be scheduled in parallel; e.g., the same person cannot attend two meetings at the same time. Thus, the scheduler sequentially allocates the non overlapping tasks.

The temporal reasoner, given the current schedule, the constraints imposed on the tasks and a problem to be solved (e.g., adding a task to the schedule or moving a task to a different time), searches for safe reallocation hypotheses concerning the problematic task. For this purpose, the execution of other tasks might be shifted back or ahead, within their start time windows, in order to reserve enough free time for it.

5 Mixed-Initiative Scheduling as a TCSP

5.1 Temporal Constraint Satisfaction Problems

As described later on in section 5.2, the constraints that must be satisfied by the tasks in a user's calendar can be represented as a Temporal Constraint Satisfaction Problem (TCSP) [19]. We thus briefly introduce this concept.

TCSPs are a class of Constraint Satisfaction Problems (CSPs) [20] tailored to the representation of temporal constraints.

A TCSP involves a set of variables $X_1, \ldots, X_n$ with continuous domains representing time points. Constraints can be unary or binary; a unary constraint:

$$(a_1 \leq X_i \leq b_1) \vee \ldots \vee (a_n \leq X_i \leq b_n)$$

constrains the value of one variable X_i to be in one of the intervals $[a_1, b_1]$, …, $[a_n, b_n]$. A binary constraint:

$$(a_1 \leq X_j - X_i \leq b_1) \vee \ldots \vee (a_n \leq X_j - X_i \leq b_n)$$

constrains the difference between two variables X_j, X_i to be in one of the intervals $[a_1, b_1]$, …, $[a_n, b_n]$.

As we shall see, TCSPs have the expressive power to capture all of the constraints of interest to this work. We solve TCSPs with a Constraint Logic Programming (CLP) solver; see Section 5.3.

For implementing some important features of our approach, we have to focus on a subclass of TCSPs, the Simple Temporal Problems (STPs) [19], where all of the constraints are binary and do not contain disjunctions:

$$(a \leq X_j - X_i \leq b)$$

This class of problems can be represented as a graph named Simple Temporal Network (STN) and has two important characteristics:

- checking whether a STN is consistent takes polynomial time [19,21]
- the same polynomial algorithm used for checking the consistency, also *minimizes* the STN; i.e., for each pair of variables X_i, X_j, it computes an interval $[a_{min}, b_{min}]$ such that in every global solution, the following holds:

  $$a_{min} \leq X_j - X_i \leq b_{min}$$

  and, vice versa, for each value $\delta \in [a_{min}, b_{min}]$ there is a global solution such that $X_j - X_i = \delta$.

We solve STPs with a specialized STN solver, as described in section 5.4.

5.2 Task Representation

We express the time constraints in the user's calendar as TCSP constraints. Starting from the basic constraints defined for a task (earliest starting time, duration, deadline, etc.), we associate two numeric variables T_s and T_e to the start and end time of each task T. For simplicity, we assume that the value of a variable T_s (resp. T_e) is the number of one-hour slots in our calendar between Monday 8.00 and the start (respectively the end) of task T.

Let us start by considering *deadlines*, *durations* and *precedences*, following the example schedule shown in Figure 1.

A *deadline*, such as "the Library meeting (LM) must take place before 11.00 on Thursday", is expressed as:

$$LM_e \leq 39$$

since in our calendar there are 39 one-hour slots between Monday 8.00 and Thursday 11.00 (see Figure 1). With a slight abuse, we use the term deadline also to indicate constraints on the exact end of a task; for example, the fact that the Prog2 class (P2) must end *exactly* on Wednesday at 13.00, is captured by:

$$P2_e = 29$$

which is equivalent to $29 \leq P2_e \leq 29$.

To express a *duration*, such as the fact that the Library meeting lasts 2 time slots, we simply write:

$$LM_e - LM_s = 2$$

A *precedence*, such as the fact that the Phone call to Mr. Smith (CS) must take place after the Ph.D meeting (PM), is expressed as:

$$CS_s - PM_e \geq 0$$

It is easy to see that all of the above constraints can be represented not only as a TCSP, but also as a Simple Temporal Network. However, there is an additional kind of constraints that is fundamental for computing an admissible schedule: the *non-overlapping* constraints. A typical non-overlapping constraint states that a task T cannot overlap with another task. For example, the fact that the Library meeting (LM) cannot overlap with the Prog2 class (P2) is expressed as:

$$P2_s - LM_e \geq 0 \vee LM_s - P2_e \geq 0$$

i.e., either LM ends before P2 starts, or vice versa. Clearly, there must be one of these constraints between each pair of non-overlapping tasks T, T' in the calendar.

There may be additional non-overlapping constraints. For example, in our scenario of section 3, the Library meeting must not take place at lunch time (i.e. from 13.00 to 14.00), nor after 17.00. In order to express this constraint on Monday, we write:

$$\begin{array}{l} LM_e \leq 5 \vee LM_e \geq 8 \\ LM_e \leq 9 \vee LM_e \geq 14 \end{array}$$

similar constraints must be added for each day of the week under consideration.

5.3 The Scheduling Module

Given the set of tasks to be allocated in the user's calendar, the scheduling module generates a schedule by handling the task definitions as constraints to be solved in a Constraint Satisfaction Problem. This type of activity has been largely explored in the research on Constraint Satisfaction; thus, we briefly describe it, leaving space for the temporal reasoning process, which is peculiar of our work.

If a task is not a precise appointment, its start and end times are time windows during which the task has to be executed (unless its duration is the same as the distance between such time points). The scheduling module thus represents the start and end time of each task as time intervals themselves, defining them as Finite Domain Variables whose domains represent the eligible time instants for starting/ending the task. For instance, if a task T must start after t0, end by t1 and its duration is d, its starting time window is [t0, t1-d].

Given the start and end Finite Domain Variables of the tasks to be scheduled and the existing non-overlapping constraints, the scheduling module performs a domain reduction on such variables in order to restrict their domains to the feasible values. If a solution exists (i.e., for each Finite Domain Variable, the domain is not null), the scheduling module explores the solution space for setting such variables to specific values, which represent the proposed allocation times. Otherwise, the scheduling module returns a "no solution" value, which describes the fact that the set of considered constraints is not satisfiable, i.e., a schedule addressing all the requirements specified by the user cannot be generated.

Different strategies could be applied in the exploration of the solution space, leading to different schedules. We selected two sample strategies to start with: allocating earlier tasks, or more urgent ones, before the others. Technically, such policies are implemented by selecting the order of the variables to be set during the exploration of the solution space (i.e., the set of possible configurations of the variables). In the *Schedule start* policy, the variables having the smallest minimum values in their domains are set before the others, which results in an early allocation of the tasks that can start earlier. In the *Schedule end* policy the variables having the smallest maximum values in their domains are set before the others, which results in an early allocation of the tasks that must end earlier.

In order to support an incremental mixed-initiative scheduling of tasks, and the possibility of reasoning on a subset of all the tasks to be considered, the scheduling module operates on a constraint set that is a clone of the original task specification. In this way, at each instant of time, the set of constraints to be considered can be reset or modified as needed. It is thus possible to create a history of the generated scheduling solutions and allow the user to navigate it and choose the preferred alternative.

It should be noticed that the constraints to be solved in the generation of a schedule might concern personal and shared tasks. Scheduling a shared task means allocating it in the calendar of all the involved actors. The scheduling module fully supports the allocation of shared tasks because the constraints belonging to the calendars of the involved users can be fused to search for a global solution by merging their constraints: in fact, even though each actor is committed to several tasks, those to be performed by different actors can overlap in the overall schedule; thus, task constraints can be merged to represent the complete set of activities to be scheduled.[1] If the overall set of constraints is not satisfiable (because there is no free slot where the involved actors can perform the task), the scheduling module returns a "no solution". However, if the failure is returned after the user has selected one of the (conservative) suggestions made by the temporal reasoner (see next section), it may still be possible to request a complete (non conservative) re-scheduling of all of the activities, to see if different solutions can be found which accommodate the new task.

5.4 The Temporal Reasoner

The deadlines, durations and precedences can be straightforwardly expressed without disjunctions, and therefore can be encoded in a Simple Temporal Network (STN). Figure 3 depicts the STN for the constraints of our running example regarding the Wednesday tasks. It shows the new task Meet Plumber (MP) to be inserted, as well as Library meeting (LM), Prog2 class (P2), Ph.D meeting (PM) and Phone Call to Smith (CS) (assuming that the tasks cannot be performed before Wednesday).

The z time point represents Monday 8.00, while the intervals on the arcs express the minimum and maximum distance between the connected time points; for example, interval $[26,39]$ on the arc connecting z and LM_e represents:

$$26 \leq LM_e - z \leq 39$$

i.e., LM must end at most on Thursday 11.00, and at least on Wednesday 10.00. The dotted arc represents the precedence between PM and CS; its associated interval (omitted for readability) would be $[0,+\infty]$; i.e., CS_s must follow PM_e by at least 0 hours.

[1] If more than one actor is involved in a task to be re-scheduled, the task instances present in the various calendars are unified by imposing that their start and end times are equal.

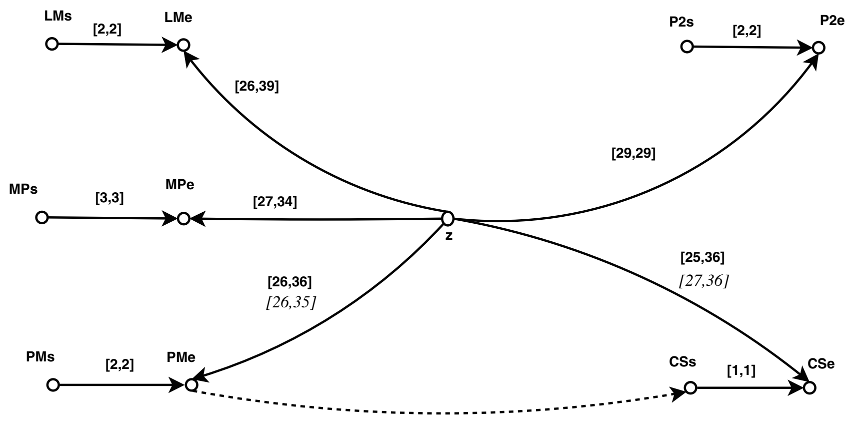


Fig. 3. Portion of the STN representing the basic constraints for the user tasks

The minimization of this STN only restricts the intervals of the arcs $z - PM_e$ and $z - CS_e$ (the restricted intervals are depicted in italics in the figure). In particular, the maximum value of PM_e (end of Ph.D meeting) becomes 35 (Wednesday 19.00) because there must be time for making the phone call to Mr. Smith afterwards. Similarly, the minimum value of CS_e becomes 27 (Wednesday 11.00) because there must be time for the Ph.D meeting before.

Note that this STN does not take the non-overlapping constraints into account and, in particular, its minimization does not affect the interval for the end PM_e of the new task (Meet plumber), which is still between 11.00 and 18.00 on Wednesday. Unfortunately, not all of the time points within this interval are admissible, as can be seen by considering, e.g., that the two time slots between 11.00 and 13.00 are rigidly allocated to the Prog2 class.

When the STN solver is invoked to show all the feasible time intervals for starting the Meet plumber task, we want it to return only admissible time points. Let us start by considering how we can take into account the non-overlapping between tasks (below, we will also discuss the non-overlapping between a task and certain time slots, such as lunch time for the Library meeting).

From the current schedule (Figure 1), we can infer the current order of the tasks that are already in the calendar. We make the assumption that the order of these tasks cannot change, while Meet plumber can be placed between any two of them.

Algorithm 1 implements this idea. It takes as inputs a new task T to insert, the sequence of the other tasks $(T_1, \dots, T_k)$ in the order in which they appear in the current schedule, and an STN $\mathcal{N}$ encoding the basic deadline, duration and precedence constraints for $T_1, \dots, T_k$ and T. Each possible positioning of T in the sequence determines a total order *Ord* among the tasks, including T). Such a total order is asserted as a set of precedence constraints into $\mathcal{N}$, and the resulting net is minimized, yielding an interval $[min_i, max_i]$ of possible values for the start T_s of T.

The algorithm returns a set *FInt* containing all of such intervals. If the current order of the tasks is not allowed to change, the intervals in *FInt* represent all of the possible choices for starting T.

Algorithm 1. Feasible intervals for adding a new task

```
input:
    new task $T$
    other tasks in current schedule order $(T_1, \ldots, T_k)$
    STN $\mathcal{N}$ (deadlines, durations, precedences)
$FInt \Leftarrow \emptyset$
for $i = 0 \ldots k$ do
    $Ord \Leftarrow (T_1, \ldots, T_i, T, T_{i+1}, \ldots T_k)$
    $\mathcal{N}' \Leftarrow$ assert order $Ord$ in $\mathcal{N}$
    $\mathcal{N}' \Leftarrow$ minimize $\mathcal{N}'$
    $FInt \Leftarrow FInt \cup$
            { get interval $[min_i, max_i]$ for $T_s$ from $\mathcal{N}'$}
end for
return $FInt$
```

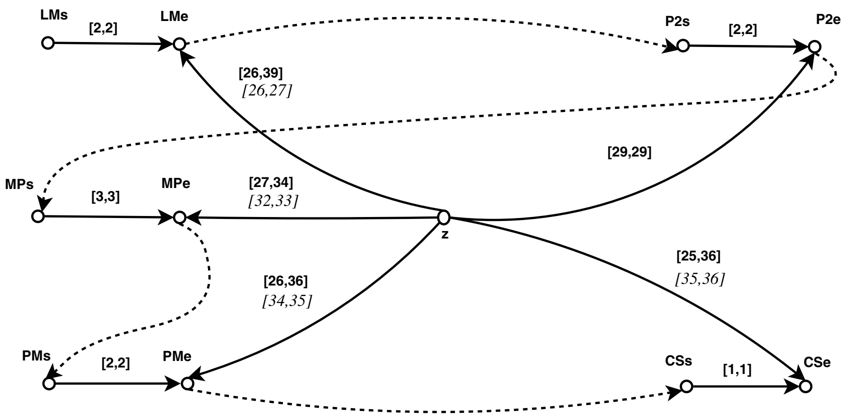


Fig. 4. STN for a specific task order

Going back to our example scenario, the new task is *MP*, the other tasks in the current scheduled order are $(LM, P2, PM, CS)$, and the basic STN $\mathcal{N}$ is the one depicted in Figure 3. Figure 4 shows the net $\mathcal{N}'$ computed by the algorithm at the 3rd iteration ($i = 2$), when the new task *MP* is placed between *P2* and *PM*.

First of all, several intervals are restricted due to the minimization. In particular, the interval on the arc $z - MP_e$ is restricted to $[32, 33]$ (Wednesday from 16.00 to 17.00); when we ask the STN for the interval on the arc $z - MP_s$, we get $[29, 30]$ meaning that, if MP is placed between *P2* and *PM*, it can start on Wednesday between 13.00 and 14.00.

The full execution of the algorithm yields the following set of intervals:

$\emptyset$	when *MP* is the first task
$\emptyset$	when *MP* is between LM and P2
$[29, 30]$	when *MP* is between P2 and PM
$[31, 31]$	when *MP* is between PM and CS
$\emptyset$	when *MP* is the last task

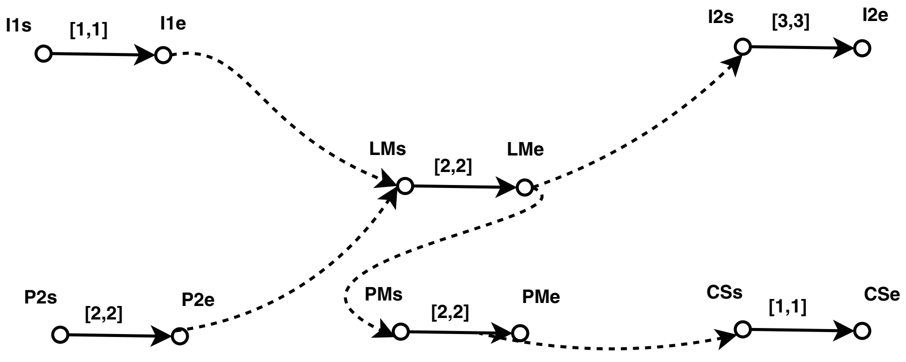


Fig. 5. STN for a specific task order with forbidden time slots

If we take the union of the overlapping intervals, we conclude that the meeting with the plumber can start on Wednesday from 13.00 to 15.00 (interval [29,31]).

Let us now show how the non-overlap constraint between a task and certain time slots can be handled by assuming that the user is not satisfied with the interval [29,31] for PM_s computed by Algorithm 1, because it would be preferable to meet the plumber in the morning. The user then attempts to make room for MP by selecting the Library meeting task and asking the STN solver to suggest where to move this task.

The computation of the possible start intervals of LM is made with an algorithm similar to Algorithm 1, which explores the effects of placing LM in each position within the current order of the other tasks $(P2, PM, CS)$. However, there is a parallel ordering to be explored; if we denote respectively as I1, I2 the lunch time (13.00 to 14.00) and the late afternoon (17.00 to 20.00) on Wednesday, the algorithm must also explore the placement of LM in each position within the order $(I1, I2)$. Figure 5 shows a portion of the STN where LM has been placed between P2 and PM in the order of tasks, and between I1 and I2 in the order of non-admissible slots.

The minimization of this particular network yields an interval [30,31] for starting LM (14.00 to 15.00). The algorithm also explores all the other combinations of the position of LM in the tasks order and in the non-admissible slots order, yielding the following admissible starting intervals:

$[24,25]$	LM first task before lunch
$[30,31]$	LM between P2 and PM after lunch
$[31,31]$	LM between PM and CS after lunch
$[36,37]$	LM on Thursday after Write paper

If we take the union of the overlapping intervals, we conclude that we can start LM on Wednesday from 8.00 to 9.00 or from 14.00 to 15.00, and on Thursday from 8.00 to 9.00. Going back to the user's goal, the Library meeting can be moved to Wednesday afternoon or Thursday morning, saving room for task Meet plumber on Wednesday morning.

It should be noticed that the described techniques could be extended to handle shared tasks. For example, after computing the slots where the Library meeting could be placed, we have only taken the current user's calendar into account. However, it might

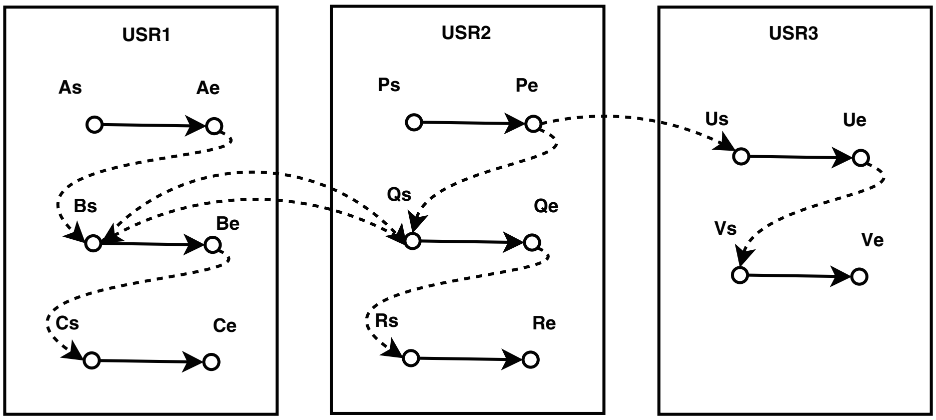


Fig. 6. Sample scenario involving multiple actors

be necessary or desirable to also take into account the calendars of the other people attending the meeting.

It is out of the scope of this paper to present the extended techniques that address this issue. In Figure 6 we give a hint of how a portion of STN encompassing multiple calendars may look like. There, tasks B and Q of USR1 and USR2 represent the same, shared task (e.g., a meeting), and this is expressed in the net by the fact that $B_s - Q_s \geq 0$ and $Q_s - B_s \geq 0$, i.e. $B_s = Q_s$. Moreover, task U of USR3 must follow task P of USR2, e.g., because the output of task P is an input for task U; this is expressed in the net by the link $U_s - P_e \geq 0$. The extensions needed to handle such an STN may benefit from distributed solving of the Simple Temporal Problem [22] and may involve negotiations among the schedules of different users (as partially done in [13], where a user affected by a change made by another user can accept or refuse it).

5.5 Technical Details

We implemented our mixed-initiative scheduler as a Java Web application. The scheduling module has been developed by integrating the JaCoP Constraint Solver [23], while the temporal reasoner has been implemented in Perl using the Graph.pm extension module [24] for representing and manipulating STNs; in particular, the minimization of the STNs is performed by invoking the implementation of the Floyd-Warshall algorithm included in Graph.pm.

The Java Web application calls the temporal reasoner as a local REST (REpresentational State Transfer) service via the HTTP protocol. The user interface is developed as a rich interface, based on the Google Web Toolkit (GWT [25]).

6 Conclusions

Calendar scheduling is a complex type of activity, which can dramatically benefit from automated support, especially in tight schedules where the allocation of new events and tasks can generate non-trivial timing conflicts to be addressed. At the same time, it cannot be reduced to the identification of solutions satisfying all the existing constraints: in fact, users handle scheduling problems in a very personal way, taking into account individual preferences, as well as information about the other involved participants. Moreover, any change which drastically modifies an existing schedule might be confusing for the involved actors because it would strongly change their daily plans. For such reasons, calendar management should support the user in finding solutions which are under her/his control and have schedule stability as a priority.

The work described in this paper follows this concept: we described a mixed-initiative calendar scheduler which proposes safe schedules and suggests conservative revisions on demand. Our system helps the user to solve scheduling conflicts but also to reschedule specific events by suggesting the time slots where they might be allocated without breaking any existing temporal constraints in the respect of the commitments of the other involved actors.

At the current stage, our mixed-initiative scheduler handles personal tasks but it only partially supports the management of shared tasks: while it generates global schedules if at least one solution satisfying the overall set of constraints exists, it cannot repair scheduling failures. In our future work we will extend the system to deal with such situations by improving the temporal reasoner that supports task re-allocation and by developing an interaction protocol that helps the involved users to reach an agreement on schedule modifications. Our future work will also be devoted to testing the scalability of our scheduler and its usability with end-users, as well as to developing its user interface for mobile access.

References

1. Grimes, A., Brush, A.: Life scheduling to support multiple social roles. In: Proceedings of CHI 2008, Florence, Italy, pp. 821–824. ACM (2008)
2. Berry, P., Gervasio, M., Peintner, B., Yorke-Smith, N.: PTIME: Personalized assistance for calendaring. ACM Transactions on Intelligent Systems 2, 40:1–40:22 (2011)
3. Ardissono, L., Petrone, G., Torta, G., Segnan, M.: Mixed-initiative scheduling of tasks in user collaboration. In: Proc. of WEBIST 2012 - Eight International Conference on Web Information Systems and Technologies, Porto, Portugal, pp. 342–351 (2012)
4. Remember The Milk: The best way to manage your tasks. never forget the milk (or anything else) again (2011), `http://www.rememberthemilk.com/`
5. Google: Google calendar (2010), `https://www.google.com/accounts/ServiceLogin?service=cl`
6. Bi101: Resource central - simplified resource management for meeting & event planners (2012), `http://www.bi101.com/solutions/resource-central/`
7. Cultured Code: Things Mac (2011), `http://culturedcode.com/things/`
8. DoIt.im: Doit anywhere, any time! (2011), `http://www.doit.im/`
9. Standss: Standss smart schedules for outlook (2012), `http://www.standss.com/smartschedules/default.asp`
10. Senkul, P., Toroslu, I.: An architecture for workflow scheduling under resource allocation constraints. Information Systems 30, 399–422 (2005)
11. Wurtz, J.: Constraint-based scheduling in oz. Selected Papers of the Symp. on Operational Research, pp. 218–223 (1996)

12. Eder, J., Pichler, H., Gruber, W., Ninaus, M.: Personal Schedules for Workflow Systems. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 216–231. Springer, Heidelberg (2003)
13. Tarumi, H., Kida, K., Ishiguro, Y., Yoshifu, K., Asakura, T.: WorkWeb system - multi-workflow management with a multi-agent system. In: Proc. of Int. ACM SIGGROUP Conference on Supporting Group Work, New York, NY, pp. 299–308 (1997)
14. Berry, P., Moffitt, M., Peintner, B., Yorke-Smith, N.: The design of a user-centric scheduling system for multifaceted real-world problems. In: Proc. of ICAPS 2007 Workshop on Moving Planning and Scheduling Systems into the Real World, Providence, RI (2007)
15. Horvitz, E., Koch, P., Subramani, M.: Mobile Opportunistic Planning: Methods and Models. In: Conati, C., McCoy, K., Paliouras, G. (eds.) UM 2007. LNCS (LNAI), vol. 4511, pp. 228–237. Springer, Heidelberg (2007)
16. Ardissono, L., Bosio, G., Goy, A., Petrone, G., Segnan, M., Torretta, F.: Collaboration support for activity management in a personal cloud. International Journal of Distributed Systems and Technologies 2, 30–43 (2011)
17. Ardissono, L., Bosio, G., Goy, A., Petrone, G., Segnan, M.: Integration of cloud services for web collaboration: A user-centered perspective. In: Models for Capitalizing on Web Engineering Advancements: Trends and Discoveries, pp. 1–19. IGI Global (2012)
18. van der Aalst, W., Ter Hofstede, A., Kiepuszewski, B., Barros, A.: Conformance checking of service behavior. ACM Transactions on Internet Technology (TOIT), Special Issue on Service-oriented Computing 8, art. 13 (2008)
19. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence 49, 61–95 (1991)
20. Dechter, R.: Constraint networks (survey). In: Encyclopedia of Artificial Intelligence, 2nd edn. John Wiley & Sons (1992)
21. Planken, L., de Weerdt, M., van der Krogt, R.: Computing all-pairs shortest paths by leveraging low treewidth. In: Proceedings of the 21st Int. Conf. on Automated Planning and Scheduling, ICAPS 2011 (2011)
22. Boerkoel, J., Durfee, E.: A comparison of algorithms for solving the multiagent simple temporal problem. In: Proceedings of the 20th Int. Conf. on Automated Planning and Scheduling, ICAPS 2010 (2010)
23. JaCoP: JaCoP - Java Constraint Programming solver (2011), `http://www.jacop.eu/`
24. Hietaniemi, J.: Graph-0.94 (2010), `http://search.cpan.org/~jhi/Graph-0.94/`
25. Google: Google Web Toolkit (2010), `http://code.google.com/intl/it-IT/webtoolkit/`