Defining a Task's Temporal Domain for Intelligent Calendar Applications

Anastasios Alexiadis and Ioannis Refanidis

Department of Applied Informatics, University of Macedonia
Thessaloniki, Greece

talex@java.uom.gr, yrefanid@uom.gr

Abstract Intelligent calendar assistants have many years ago attracted researchers from the areas of scheduling, machine learning and human computer interaction. However, all efforts have concentrated on automating the meeting scheduling process, leaving personal tasks to be decided manually by the user. Recently, an attempt to automate scheduling personal tasks within an electronic calendar application resulted in the deployment of a system called SELFPLANNER. The system allows the user to define tasks with duration, temporal domain and other attributes, and then automatically accommodates them within her schedule by employing constraint satisfaction algorithms. Both at the design phase and while using the system, it has been made clear that the main bottleneck in its use is the definition of a task's temporal domain. To alleviate this problem, a new approach based on a combination of template application and manual editing has been designed. This paper presents the design choices underlying temporal domain definition in SELFPLANNER and some computational problems that we had to deal with.

1 Introduction

Electronic calendar organizers constitute inseparable companions of almost every busy person, such as managers, professionals, academics, politicians and others. Microsoft Outlook, Google Calendar and Yahoo! Calendar constitute mainstream products of the software industry, with new features being constantly added to them. These products provide convenient ways to help the user organize her tasks as well as to arrange meetings with others (based usually on message exchanging). On the other hand, several researchers in the last decade concentrated on embedding intelligence in electronic organizers, focusing on automating the meeting scheduling process either by exchanging messages between the potential participants or by learning a user's preferences ([1, 4, 7]).

Recently, a research effort attempting to enhance the intelligence of electronic organizers by automating the scheduling of personal task procedure resulted in the deployment of a system called SELFPLANNER [5]. Contrary to other intelligent calendar application efforts that ignore tasks or retain them in task lists without any

attempt to put them on the user's calendar [2], SELFPLANNER puts them into the user's calendar, taking into account several types of constraints and preferences.

Perhaps the most important attribute of a task is its temporal domain, i.e. when the task can be executed. However, defining a temporal domain is also the most cumbersome part of a task's definition. Having recognized that from the system's design phase, we devised several mechanisms to facilitate domain definition. They are mainly based on a combination of template application and manual editing. So, this paper concentrates on the domain definition issue both from a human-computer interaction and from an algorithmic point of view.

The rest of the paper is structured as follows: Section 2 highlights the key features of the SELFPLANNER application. Section 3 presents the internal representation of task domains in SELFPLANNER, whereas Section 4 discusses the algorithmic issues incurred by the way domains are represented. Finally, Section 5 concludes the paper and poses future research directions.

2 SELFPLANNER Overview

SELFPLANNER is a web-based intelligent calendar application that helps the user to schedule her personal tasks (Fig. 1). With the term 'personal task' we mean any task that has to be performed by the user and requires some of her time. Meetings are considered as milestones, i.e. they cannot be moved by the system. The user can schedule her meetings using alternative tools, such as Google Calendar meeting arrangement facilities.

Each task is characterized by its duration and its temporal domain [6]. A domain consists of a set of intervals, where the task can be scheduled. A task might be interruptible, i.e. it can be executed in parts with the sum of the durations of these parts being equal to the task's duration. Several constraints on the durations of these parts and their temporal distances can be defined. A task can also be periodic, i.e. it has to be performed many times, e.g. daily or weekly. A location or a set of locations is attached to each task; to execute a task or a part of it, the user has to be in one of these locations. Travelling time between pairs of locations are taken into account when the system schedules adjacent tasks.

Ordering constraints and unary preferences are also supported by the system. Unary preferences are monotonic linear or step utility functions over the temporal domain of a task, denoting when the user prefers the task to be scheduled. SELFPLANNER uses an adaptation of the Squeaky Wheel Optimization framework [3] to solve the resulting scheduling problem, while trying to optimize the sum of the various preferences.

SELFPLANNER utilizes Google Calendar for presenting the calendar to the user, and a Google Maps application to define locations and compute the time the user needs to go from one location to another. The system is available for public use at http://selfplanner.uom.gr.

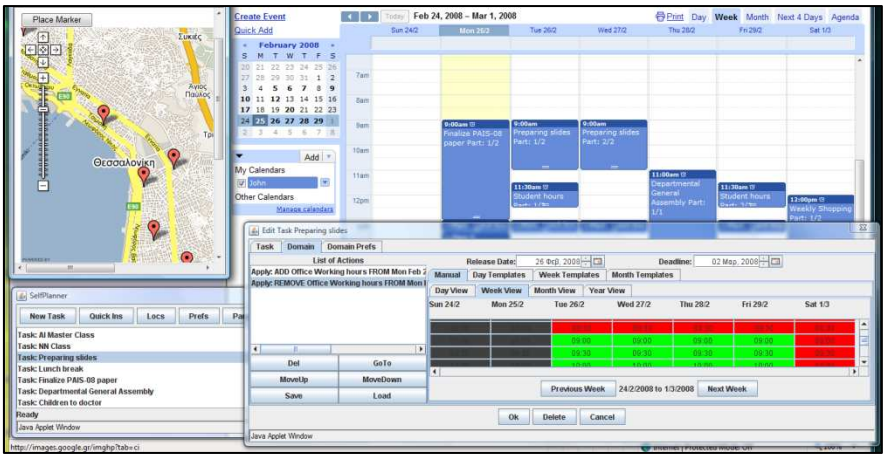

Fig. 1. Overview of the SELFPLANNER intelligent calendar application.

3 Domain Representation

From a theoretical point of view, the temporal domain of a task consists of a set of intervals. For practical reasons we consider only integer domains. In the context of the application, a unit corresponds to the quantum of time that, as in most electronic calendars, is 30 minutes. For reasons of clarity, in the following we will use a notation of the form ⟨DD/MM/YY HH:MM⟩ to denote time points. Depending on the context, several parts of the time stamp will be omitted or altered.

Using intervals to represent domains is however problematic both from a computational and from a user's experience point of view. Suppose for example that a user wants to schedule a task of 2 hours duration and this task has to be performed during office hours next week. Supposing a 5 days working week, this might results in five intervals of the form, say:

[⟨27/10/08 09:00⟩, ⟨27/10/08 17:00⟩] … [⟨31/10/08 09:00⟩, ⟨31/10/08 17:00⟩]

Imagine now what happens if the same task has a deadline after a month or a year: Storing and retrieving the domain of this task would be a time- and space-consuming process. Even worse, having the user to define this domain would be an inhibitory factor to use the system at all. To overcome these deficiencies, we selected to avoid using interval representation for temporal domains.

3.1 Templates and List of Actions

A template is a pattern with specific duration and with no absolute time reference. SELFPLANNER supports three types of templates: Daily, Weekly and Monthly.

Each template consists of a set of intervals covering the entire pattern's period and denoting which slots are allowed for a task's execution (the remaining are not). For example, a daily template of lunch hours would comprise a single interval, say [⟨12:30⟩, ⟨15:00⟩]. Similarly, a weekly template of office hours would comprise five intervals of the form [⟨Mo 09:00⟩, ⟨Mo 17:00⟩] to [⟨Fri 09:00⟩, ⟨Fri 17:00⟩]. Note that the daily template's interval does not have any day reference, whereas the weekly template's intervals have a relative reference of the week's day.

Templates can be used to define domains. The simplest way is to combine a template with a release date and a deadline. However, to increase flexibility in domain definition through templates, we distinguish four different ways in applying a template. These are the following:

- *Add included*, denoted with ⇧: The time slots identified by the template are added in a task's temporal domain.
- *Remove excluded*, denoted with ⥥: The time slots not identified by the template are removed from a task's temporal domain.
- *Add excluded*, denoted with ⥣: The time slots not identified by the template are added in a task's temporal domain.
- *Remove included*, denoted with ⇩: The time slots identified by the template are removed from a task's temporal domain.

As an example, consider again the daily lunch hours template, named *Lunch*. If we want to include lunch hours to a task's domain we use ⇧*Lunch*. If we want to say that a task is to be executed only during the lunch hours, we use ⇧⥥*Lunch*. If we want to exclude lunch hours from a task's domain we use ⇩*Lunch*.

3.2 List of Actions and Temporal Domains

A temporal domain can be defined through a sequence of template applications. For example, one task's domain might consist of office working hours excluding the lunch hours. This could be defined by a list of actions, that initially adds office hours to the temporal domain and then removes lunch hours.

More formally, a *domain action* is defined as a temporally constrained template application. This is denoted by providing an absolute interval with the template, e.g. ⇧⥥*Lunch*@[⟨27/10/08 00:00⟩, ⟨28/10/08 00:00⟩]. Domain actions that are not temporally constrained apply to the whole task's domain.

A *list of domain actions* is an ordered sequence of them. For example, the following list defines office working hours excluding the lunch hours (note that domain actions are not temporally constrained, so they apply to the whole task's domain):

⇧⥥*OfficeHours*,
⇩*Lunch*

Manual editing a domain, i.e. adding or removing a time slot without the use of a template, can be seen as a special case of template application. Suppose we have

a daily template named *All*, consisting of the single interval [⟨00:00⟩, ⟨24:00⟩], i.e. the whole day (operators ⏛ and ⏜ are meaningless for this template). So, manually adding (removing) an interval to (from) a task's temporal domain is equivalent to applying ⇧*All* (⇩*All*) temporally constrained over this interval.

Finally, a *temporal domain* is defined as a list of domain actions, accompanied by a release date and deadline. So, the following specifies a task's domain over the week from 27/10/08 to 31/10/2008, including all office hours but the lunch hours:

[⟨27/10/08 00:00⟩, ⟨31/10/08 24:00⟩]
⇧⏛*OfficeHours*
⇩*Lunch*

The semantics of a domain are the following:

1. All time slots before the release date or after the deadline are excluded from the domain.
2. A time slot is included in the domain, if there is a domain action that adds this time slot in the domain, whereas no subsequent domain action removes the time slot.
3. A time slot is excluded from the domain, if there is a domain action that removes this time slot, whereas no subsequent domain action adds the time slot.
4. All unspecified time slots are considered as included in the domain.

4 Computational Issues

Using lists of domain actions to represent temporal domains gives rise to interesting computational problems, such as whether a time slot is included in the domain or not, how to transform the domain into the traditional representation with list of intervals or, finally, how to simplify the list of domain actions. These issues are treated in the following subsections.

4.1 Domain Inclusion

Knowing whether a time slot is included in a task's temporal domain or not is important, among others, when graphically displaying parts of the domain on the screen. The following algorithm answers this question:

Algorithm 1. *GetTimeSlotStatus*

Inputs: A domain represented by a list of domain actions and a time slot *T*.
Output: Either of the *included* or *excluded* values.

1. If *T* is before the release date or after the deadline, return *excluded*.
2. Let *D* be the last domain action. If no such action exists, then *D* is NULL.
3. While *D*≠*NULL*

   a. If *D* adds the *T*, return *included*.
   b. If *D* removes *T*, return *excluded*.
   c. Let *D* be the previous domain action. If no such action exists, *D* is *NULL*.
4. Return *included*.

Algorithm *GetTimeSlotStatus* is very fast. Indeed, suppose that the action list has *N* entries and each template has at most *M* intervals, then the worst case complexity is O(*N*·*M*), with *N* and *M* usually taking small values.

4.2 List of Intervals

It is often required to transform a temporal domain represented by a list of domain actions to the traditional list of intervals. For example, most existing schedulers do not support the list of domain actions representation. So, algorithm *GetIntervals* is a generalization of algorithm *GetTimeSlotStatus* and does exactly that:

Algorithm 2. *GetIntervals*

Inputs: A domain represented by a list of domain actions.
Output: A list of intervals.

1. Let **A** be a table of integers, whose size equals the number of time slots between the domain's release date and the deadline. Let *S* this size. Initialize **A** with zeroes. Let *C*=0.
2. Let *D* be the last domain action. If no such action exists, then *D* is *NULL*.
3. While *D*≠*NULL* and *C*<*S*.
   a. For each time slot *T* added by *D*
      i. If **A**[*T*] is 0, then set **A**[*T*] to 1 and increase *C* by 1.
   b. For each time slot *T* removed by *D*
      i. If **A**[*T*] is 0, then set **A**[*T*] to -1 and increase *C* by 1.
   c. Let *D* be the previous domain action. If no such action exists, *D* is *NULL*.
4. If *C*<*S*
   a. For each time slot *T* such that **A**[*T*]=0, set **A**[*T*]=1.
5. Create a list of intervals by joining consecutive time slots having **A**[*T*]≥0.

Algorithm 2 is very fast, however it might have significant memory requirements in case of large domains due to the definition of the temporary variable **A**. However, an alternative design that would directly encode the new domain in intervals would be inefficient, since step 3 would require to traverse the entire list of intervals to decide whether a time slot has already got a status or not.

4.3 Simplifying Domains

There are cases where several domain actions can be removed from the list with-

out any change in the resulting domain. For example, suppose ⇧*OfficeHours* exists in a domain action list. This domain action adds to the domain all the included template's intervals and, at the same time, it removes all excluded template's intervals. Furthermore, this domain action is not temporally constrained, so it covers the entire domain. In this case, any domain action occurring before this one wouldn't have any effect in the domain and thus it could be safely removed.

Detecting domain actions that can be safely removed from the action list requires a simple change in Algorithm 2. In particular, in step 3 we should check, for each domain action *D*, whether the domain action has increased *C* or not. In the latter case the domain action does not affect the temporal domain and can be removed. However, from an application point of view, this removal should (and does) not occur without prior confirmation from the user, since the user might intend to remove or modify some of the subsequent domain actions, which could result in ineffective domain actions to become effective.

5 Conclusions and Future Work

This paper presented an alternative way to represent temporal domains of tasks within a deployed intelligent calendar application. This representation can be used both for internal representation and for user interface purposes. We also presented efficient algorithms that answer questions as to whether a specific time-slot is included in the domain or what is the equivalent representation with intervals.

All these features have been implemented in a deployed intelligent calendar assistant application called SELFPLANNER. Fig. 2 shows the task's domain definition dialog box. As we can see, there are four main tabs, labelled as Manual, Day, Week and Month. The first one is for manual editing the domain, whereas the other three are for defining and applying templates. Manual editing allows also the user to view the current domain in various views. What is interesting is that the current list of actions is visible to the user at the left hand side of the dialog box. The user can not only watch it, but she can also change it, by removing domain actions or change their order. Every change in the list of domain actions is immediately displayed on the manual editing tab of the dialog box. She can also save the list of domain actions to retrieve it to define the domain of another task.

As for the future, we are working on allowing the user to link templates with the tasks. Currently, as far as a template is applied to a task's domain, any subsequent change in the template does not affect the task's domain. In other words, a copy of the template is used for each task. However, there are cases where the user might want to change the domains of several tasks that have been defined using a specific template, without the need to redefine all domains (e.g. suppose that the store hours have been changed, so all tasks concerning shopping have to change their domains). The most natural way to do that is to modify the template, but this requires that the template is not embedded but linked with the tasks.

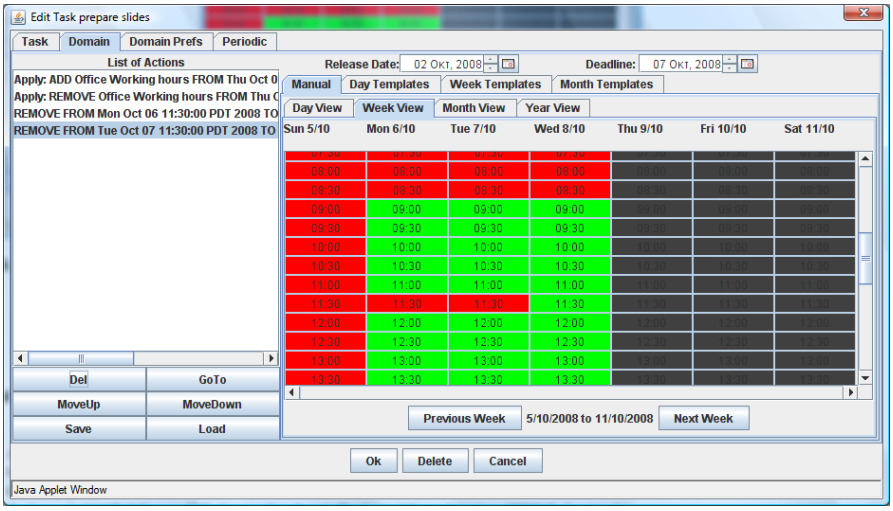

Fig. 2. Overview of the domain definition dialog box.

We are also working on defining and integrating a task ontology, where each class of tasks has its default partially defined temporal domain. To conclude with, we believe that intelligent calendar assistants will play a significant role in organizing our lives in the future but a crucial factor for their adoption is their usability. The work presented in this paper is, we hope, a step towards that direction.

References

1. Berry P, Conley K, Gervasio M, Peintner B, Uribe T & Yorke-Smith N (2006) Deploying a Personalized Time Management Agent. 5th Intl Joint Conf. on Autonomous Agents and Multi Agent Systems, Industrial Track, Hakodate, Japan, pp. 1564-1571.
2. Conley K Carpenter (2007) Towel: Towards an Intelligent To-Do List. AAAI Spring Symposium on Interaction Challenges for Artificial Assistants, Stanford, CA.
3. Joslin D E & Clements D P (1999) “Squeaky Wheel” Optimization. Journal of Artificial Intelligence Research, vol. 10: 375-397.
4. Modi P J, Veloso M, Smith S F & Oh J (2004) CMRadar: A Personal Assistant Agent for Calendar Management. Workshop on Agent Oriented Information Systems.
5. Refanidis I & Alexiadis A (2008) SelfPlanner: Planning your Time! ICAPS 2008 Workshop on Scheduling and Planning Applications, Sydney.
6. Refanidis I (2007) Managing Personal Tasks with Time Constraints and Preferences. 17th Intl. Conf. on Automated Planning and Scheduling Systems, Providence, RI.
7. Singh R (2003) RCal: An Autonomous Agent for Intelligent Distributed Meeting Scheduling. Tech. report CMU-RI-TR-03-46, Robotics Institute, Carnegie Mellon University.