# SELFPLANNER: Planning your Time!

**Ioannis Refanidis and Anastasios Alexiadis**

University of Macedonia, Dept. of Applied Informatics
Thessaloniki, Greece
yrefanid@uom.gr, talex@java.uom.gr

## Abstract

This paper presents SELFPLANNER, a web-based intelligent calendar application that helps a user plan her time. Contrary to other calendar assistants that concentrate on automating meeting scheduling, SELFPLANNER emphasizes on scheduling the personal tasks of a user, leaving meeting arrangement for external handling. The two key features of SELFPLANNER, also critical factors for its potential broader adoption, are problem modeling and user interface. SELFPLANNER supports simple, interruptible and flexible periodic tasks, arbitrary domains, constraints over the parts of an interruptible task, binary constraints and unary and binary preferences over tasks, location references and classes of locations. As for user interface, SELFPLANNER integrates with Google Calendar and a Google Maps based application, whereas it introduces an innovative way to define domains, based on manual selections and user-defined reusable templates. The core of the system is based on the Squeaky Wheel Optimization algorithm, with efficient domain dependent heuristics.

## Introduction

Modern electronic organizers, such as MS-Outlook, Google Calendar and Yahoo Calendar, do not provide for automated scheduling of a user's tasks. Users have to manually place their tasks into the calendar, as well as to arrange meetings with others. These applications provide various functionalities to assist the user to put her tasks into the calendar, detect conflicts, merge calendars, assign tasks to other users, arrange meetings, share and publish the calendar. The need for intelligent assistance to schedule a user's tasks has already been identified (Refanidis, McCluskey and Dimopoulos, 2004). It is a general impression that most of the effort in developing new editions of office applications is towards personal time management simplification. However, current status still remains behind automated scheduling abilities being incorporated into these programs.

There are several research efforts in the recent years to embed intelligence into calendar applications (Berry et. al., 2006; Modi et. al., 2004; Singh, 2003), but all of them concern meeting scheduling, with personal tasks being considered as busy time in a user's plan. However, most of a user's activity with calendar applications concerns personal tasks and, although many of them are inelastic (e.g. get the child to school, giving a lecture etc), there are many others, such as going shopping, writing a paper or breaking for lunch that, more ore less, admit of scheduling. Typically, users spend much time to organize their tasks, whereas the resulting plans are usually far from optimal.

We envision future calendar applications as really intelligent assistants that share a user's goals and elicit plans to achieve them. To realize this functionality, they need to solve a planning problem, where the initial state is the user's current status, including the status of her micro-world (need to address a frame problem), actions are retrieved from a task ontology, and goals are set by the user. Intelligent calendar assistants can be seen as a core module of a more general architecture of intelligent agents that possess our profile, collect information from several sources, learn and organize us (Myers, 2006).

This paper presents SELFPLANNER, a necessary step towards this vision. SELFPLANNER is a web-based application built on top of a fast domain-specific scheduler, that integrates several web technologies and Google-based applications. SELFPLANNER automates the elaboration and optimization of personal plans in the presence of binary constraints and preferences; on the other hand, meetings are considered as busy time, leaving users use manual procedures of Google Calendar (or third party applications) to arrange them. Keeping these two procedures apart was a decision, since allowing a new meeting to invoke rescheduling of existing tasks (potentially involving other meetings) could lead to endless loops.

Apart from its scheduling engine, the two key features of SELFPLANNER are problem modeling and user interface. Both these features are critical in order for the system to be adopted by its potential target groups, i.e. people with very tight schedules (managers, academics etc). Concerning problem modeling, each task is characterized by a set of attributes such as duration, domain, class of locations, periodicity, interruptibility etc. In case of periodic and interruptible tasks, additional information must be specified, such as the period or the min/max duration of each part of an interruptible task. Binary constraints between tasks, such as ordering and proximity ones are allowed. Similarly, unary and binary preferences can be defined.

Concerning user interface, SELFPLANNER uses innovative techniques to facilitate data entry. Perhaps the most tedious part of a task's definition concerns its domain. A domain may consist of several time intervals, distributed among long periods, e.g. shop's working hours. The user defines domains through a combination of manual selections and template applications. Templates are patterns that can be applied to long time periods. The user can create and reuse templates, as well as save her operations in order to reapply them to other tasks. Other user interface innovations concern a Google Maps based application that helps the user define locations and compute spatial and temporal distances between them; classes of locations that provide alternative places where a task can be performed; periodicity, where the occurrences of a task are not scheduled in exactly the same day/time; suggestions to relax constraints in case of failure to schedule all of them etc.

The rest of the paper is structured as follows: First we define the problem of personal tasks scheduling and then we sketch the adopted algorithmic solution. Next we present the system's architecture followed by a description of the key features of the system and some indicative use cases. Finally we present related work and conclude the paper while posing future directions.

## Underlying Model

We adopt the following problem formulation (Refanidis, 2007), which forms an abstraction of the actual SELFPLANNER problem. Time is considered a non-negative integer, with zero denoting the current time. There is a set $T$ of $N$ tasks, $T=\{T_1, T_2, \ldots, T_N\}$. Each task $T_i \in T$ is characterized by its duration $dur_i$[1]. All tasks are considered interruptible, i.e. they can split into parts to be scheduled separately. The decision variable $p_i$ denotes the number of parts in which the $i$-th task has been split, with $p_i \geq 1$. $T_{ij}$ denotes the $j$-th part of the $i$-th task, $1 \leq j \leq p_i$. For each $T_{ij}$, the decision variables $t_{ij}$ and $dur_{ij}$ denote its start time and duration. The sum of the durations of all parts of a task must equal its total duration (C1).

For each task $T_i$, the maximum and minimum allowed duration for its parts, $smax_i$ and $smin_i$ (C2), as well as the minimum allowed temporal distance between every pair of its parts, $dmin_i$ (C3), are given. Depending on the values of $smax_i$ and $smin_i$ and the overall duration of the task $dur_i$, implicit constraints are imposed on $p_i$. For example, if $smin_i > dur_i/2$, then $p_i=1$, so task $T_i$ is non-interruptible.

Each task $i$ has its given domain $D_i$, consisting of a set of intervals within which all of its parts have to be scheduled (tasks with infinite horizon of execution are not considered): $D_i=[a_{i1},b_{i1}] \cup [a_{i2},b_{i2}] \cup \ldots \cup [a_{i,Fi},b_{i,Fi}]$, where $F_i$ is the number of intervals of $D_i$ (C4). Obviously, $a_{ij}+smin_i \leq b_{ij}$ as well as $b_{ij} < a_{i,j+1}$ must hold for each $1 \leq i \leq N$, $1 \leq j \leq F_i$.

[1] In the following we use the notation $x_i$ to abbreviate $T_i.x$, where $x$ is any attribute of the task structure. In case of multiple subscripts, e.g. $x_{ij}$, the first one, i.e. $i$, indicates the task.

A set of $M$ locations, $Loc=\{L_1, L_2, \ldots, L_M\}$ and a two dimensional matrix $Dist$ (not necessarily symmetric) with their temporal distances (non-negative integers) are given. Each task $T_i$ has its own spatial references $Loc_i \subseteq Loc$, denoting alternative places where the user should be in order to execute each part of the task (the user has not to execute all the parts of a task in the same location). The decision variable $l_{ij} \in Loc_i$ denotes the particular position where $T_{ij}$ will be executed (C5, C6).

For any subset of tasks $S \subseteq T$, a constraint $c$ over these tasks may be defined, thus determining the valid ways to schedule the tasks of the set. Constraints refer only to time, not to location references; however the role of the locations in deciding when to schedule a task is important, since the decision to schedule a task at a specific location may affect the domain of other tasks. Each constraint $c$ is defined by a function $propagate_c(S)$, which, given a set of partially instantiated tasks $S$, propagates the constraint $c(S)$ over the domains of these tasks and returns returns $\perp$ if any domain remains empty, otherwise it returns $\top$ (C7).

As a simple example consider the ordering constraint over non-periodic tasks, denoted with $\prec(T_i,T_j)$, meaning that no part of the $j$-th task can start its execution until all parts of the $i$-th task have finished their execution. In this case, $propagate_{\prec}(T_i,T_j)$ applies bounds consistency (Van Hentenryck, Saraswat and Deville, 1998) to the domains of $T_i$ and $T_j$. Other constraints, including higher order ones, can be defined as well.

Finally, a set $V$ of time preferences over sets of tasks are also allowed. A preference $v \in V$ over a set of tasks $S$ is defined as a function $v{:}\prod_{Ti \in S} D_i \rightarrow \Re$, i.e. function $v$ maps each combination of the domains of the tasks of $S$ to a real number. Preference functions are usually *max*-type functions that greedily try to estimate a best-case scheduling scenario based on the current domains of the involved tasks. For example, a unary preference could return the utility of the best time-window when the task could be scheduled, whereas a binary preference of an *away* type could return the utility of the maximum possible temporal distance where the two involved tasks could be scheduled. However, other types of functions, such as *average*, can be adopted as well.

So, after these definitions, the problem of managing personal tasks can be formulated as follows:

*Problem definition*: Given a set of tasks $T$ with their attributes, a set of constraints $C$ and a set of preferences $V$, find appropriate values for the decision variables $p_i$, $t_{ij}$, $dur_{ij}$, $l_{ij}$, where $1 \leq i \leq N$, $1 \leq j \leq p_i$ such as to maximize the expression:

$$\sum_{v_k \in V} v_k(S_k) \qquad (1)$$

subject to the following constraints:

C1: $\forall i, 1 \leq i \leq N: \sum_{j=1}^{p_i} dur_{ij} = dur_i$

C2: $\forall i,j, 1 \leq i \leq N, 1 \leq j \leq p_i : smin_i \leq dur_{ij} \leq smax_i$

C3: $\forall i,j,k, 1 \leq i \leq N, 1 \leq j \leq p_i, 1 \leq k \leq p_i : j \neq k \Rightarrow$
$t_{ij} \geq t_{ik} + dur_{ik} + dmin_i \vee t_{ik} \geq t_{ij} + dur_{ij} + dmin_i$

C4: $\forall i,j, 1 \leq i \leq N, 1 \leq j \leq p_i \; \exists k, 1 \leq k \leq F_i : a_{ik} \leq t_{ij} \leq b_{ik} - dur_{ij}$

C5: $\forall i,j, 1 \leq i \leq N, 1 \leq j \leq p_i : l_{ij} \in Loc_i$

C6: $\forall i,j,m,n, 1 \leq i \leq N, 1 \leq j \leq p_i, 1 \leq m \leq N, 1 \leq n \leq p_m : i \neq m \vee j \neq n$
$\Rightarrow t_{ij} + dur_{ij} + Dist(l_{ij}, l_{mn}) \leq t_{mn} \vee t_{mn} + dur_{mn} + Dist(l_{mn}, l_{ij}) \leq t_{ij}$

C7: $\forall c(S) \in C, propagate_c(S) = \top$

## Squeaky Wheel Optimization

Squeaky Wheel Optimization (SWO) is a general optimization framework that can be adapted to several constraint satisfaction problems (Joslin and Clements, 1999). The core of SWO is a Construct/Analyze/Prioritize cycle, as shown in Figure 1(a). Constraint variables are placed in a priority queue in decreasing order of an initial estimate of the difficulty to assign a value to each one of them. A solution is constructed by a greedy algorithm, taking decisions in the order determined by the priority queue. This solution is then analyzed to find those constraint variables that were the "trouble makers". The priorities of the "trouble makers" are increased, causing the greedy constructor to deal with them sooner in the next iteration. This cycle repeats until a termination condition occurs.

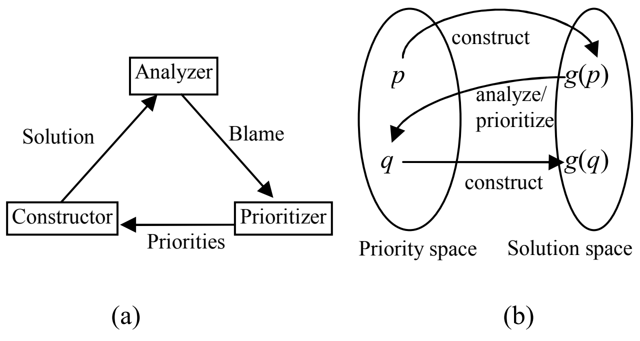


**Figure 1.** (a) The SWO cycle. (b) Coupled search spaces.

SWO is a fast but incomplete search procedure. As shown in Figure 1(b), SWO searches in two coupled spaces: The priority space and the solution space. The greedy construction algorithm defines a function *g* from the priority queues to the solutions, i.e. for each ordering *p* of the tasks a schedule *g*(*p*) is defined. However, function *g* may be neither surjective nor injective; so, many feasible solutions may not correspond to any ordering of the tasks in the queue.

We adapted SWO to SELFPLANNER using several domain dependent heuristics that measure the impact of the various ways of scheduling a specific task (including both time and location) to the remaining ones. In particular, the difficulty $diff(T_i)$ to schedule a task $T_i$ is defined as the maximum between two metrics, $m_1$ and $m_2$, which in turn are defined as follows:

Metric $m_1$ of a task $T_i$ is defined as the ratio between the total duration of the task and the net size of its domain, i.e.:

$$m_1(T_i) = dur_i / net(D_i)$$

where the net size *net*(*D*) of a domain *D* consisting of a set of intervals is defined as the sum of the widths of these intervals.

Metric $m_2$ of a task $T_i$ is defined as the ratio between the minimum possible makespan of the task and the width of its domain, i.e.:

$$m_2(T_i) = min(makespan(T_i)) / width(D_i)$$

The overall difficulty to schedule a set of tasks *S* is defined as the product of their individual difficulties:

$$overall(S) = \prod_{T_i \in S} diff(T_i)$$

So, tasks are initially placed at the queue in decreasing order of their individual difficulties, whereas each task is scheduled at the time slot where the overall difficulty to schedule the remaining tasks is minimized. For each possible time window to schedule the current task, constraint propagation is employed to compute the domains of the remaining tasks before computing the difficulty to schedule them.

In case of preferences, the ratio between overall difficulty and approximated overall utility is minimized:

$$\frac{(overall(S))^a}{\left(\sum_{v_k \in V} v_k\right)^b}$$

Experimental results have shown that SWO is significantly more efficient and effective (under time limit) than other complete search algorithms. The details of the specific adaptation of SWO to SELFPLANNER have been presented in (Refanidis, 2007).

## System Architecture

SELFPLANNER is a web based application running over a planning server (Figure 2), which implements the SWO algorithm. All data are stored centrally, so the user can access the application from any networked computer. The user edits task data using user-friendly dialog boxes. The planning server solves the scheduling problem and inserts suitable entries in the user's Google Calendar account. Finally, the user watches her calendar directly into Google Calendar. The user can also add tasks directly into her Google Calendar account; during scheduling, these tasks are considered as busy time by the system. SELFPLANNER also integrates a Google Maps based application, in order to obtain the user's list of locations, as well as to compute temporal distances between them. The Google Maps application "communicates" with the core SELFPLANNER system through shared files.

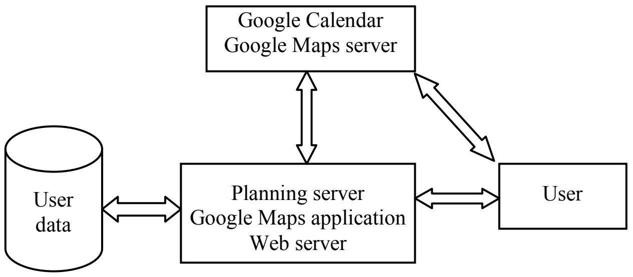


**Figure 2:** SELFPLANNER overall architecture

SELFPLANNER encompasses several technologies and platforms. The main system consists of a Java application, with the user interface being a Java applet. All connections between the user and the system are secure. The Google Maps application uses PHP and Javascript. The planning engine that implements the SWO algorithm has been implemented in C++. Finally, user data such as task details and the current plan are retained as serializable Java objects (binary files).

## SELFPLANNER Features

This section highlights the key features of SELFPLANNER from a user point of view. An overall view of the system, with the main windows open, is given in Figure 3.

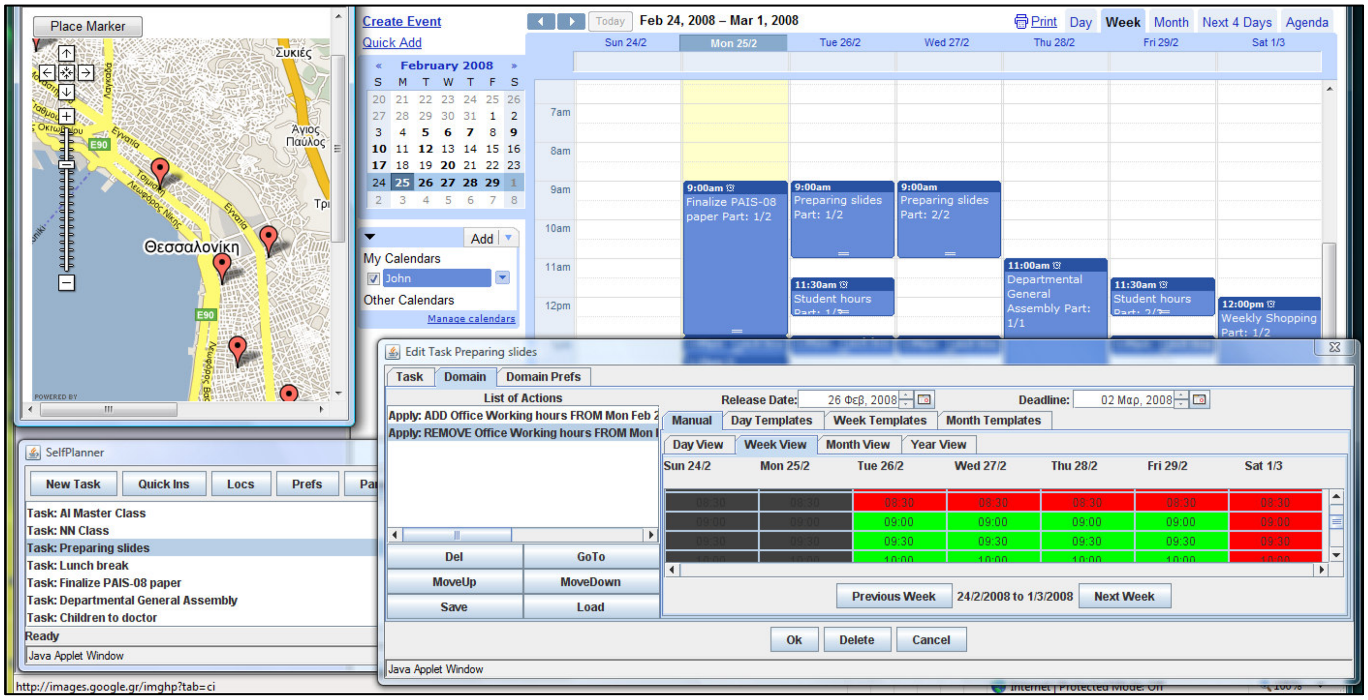
The main entity in the "planning your time" concept is the task. Using suitable user interface modules, the user can define all the parameters of a task, as they have been described in Section 2, i.e. duration, domain, interruptibility, periodicity, alternative locations, constraints and preferences.

Perhaps the most tedious job when defining a task is the definition of its domain. SELFPLANNER uses an innovative way for defining domains, based on combining manual interval selections and template applications. Manual selections allow the user to include or exclude (by clicking or dragging) specific intervals from a domain (a minimum time slot of 30 minutes is assumed). On the other hand, templates allow the user to apply the same pattern of inclusions/exclusions over long periods. A template comprises a set of *green* and *red* values characterizing time slots over a relative interval. Three types of templates are supported: daily, weekly and monthly. The user can define and store as many templates as she needs. A template can be applied over the whole domain or over part of it, in four different ways: adding/removing the *green* slots, and adding/removing the *red* slots. The user can apply several templates; however the order in which they are applied matters.

Another innovation concerns the way domains are retained in memory: they are not retained as lists of intervals (which can be difficult to handle in case of large domains) but as lists of user actions. A user action can be either a manual addition/removal of a time slot in/from the domain, or the application of a template. The list of user actions is accessible to the user, who can modify it by changing the order of the actions or delete some of them. Efficient algorithms have been developed to answer questions such as whether a particular time slot is included or not in the domain.

SELFPLANNER treats all tasks as interruptible, with non-interruptible tasks being characterized by *smin=smax=dur*. The decision in how many parts to split an interruptible task is taken by the greedy scheduling algorithm. In addition, a task may be periodic. Periodic tasks are considered as collections of simple tasks. Similarly to interruptible tasks, SELFPLANNER treats all tasks as periodic, with non-periodic tasks being characterized by a single iteration. Each periodic task has a predetermined finite number of periods. The period may be either a day, or a week, or finally a month. The end-points of each period are fixed in advance: for example, weekly periodic tasks start on Sunday and end on Saturday. However, specific intervals within a period may be out of the domain. The various instances of a periodic task are scheduled separately, so, depending on the domain of the task, they may be scheduled in different offsets within their periods. For example, the first iteration of a weekly periodic task might be scheduled on Monday, whereas the second iteration might be scheduled on Thursday. In addition, the user may ask not to schedule an instance of the task for specific periods (e.g. holiday breaks). Note that periodicity does not exclude interruptibility, i.e. a periodic task may be interruptible as well.

A task is characterized by one or more locations. In order to execute the task (or a part of it), the user must be in one of its locations. In case of interruptible and periodic tasks, different parts of the same task may be scheduled in different locations. In order to facilitate location entry, SELFPLANNER supports classes of locations: A class is a set of distinct locations and can be assigned to a task instead of a simple location. Travelling times between different locations are taken into account when scheduling tasks that are located in different places. SELFPLANNER integrates a Google Maps based application in order to obtain the user's list of locations and to compute distances between them.



Feb 24, 2008 – Mar 1, 2008

February 2008

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |

John

| | Sun 24/2 | Mon 25/2 | Tue 26/2 | Wed 27/2 | Thu 28/2 | Fri 29/2 | Sat 1/3 |
|---|---|---|---|---|---|---|---|
| 9am | | 9:00am Finalize PAIS-08 paper Part: 1/2 | 9:00am Preparing slides Part: 1/2 | 9:00am Preparing slides Part: 2/2 | | | |
| 11am | | | 11:30am Student hours Part: 1/… | | 11:00am Departmental General Assembly Part: 1/1 | 11:30am Student hours Part: 2/… | |
| 12pm | | | | | | | 12:00pm Weekly Shopping Part: 1/2 |

Task: AI Master Class
Task: NN Class
Task: Preparing slides
Task: Lunch break
Task: Finalize PAIS-08 paper
Task: Departmental General Assembly
Task: Children to doctor
Ready

Edit Task Preparing slides

List of Actions
Apply: ADD Office Working hours FROM Mon Feb 2
Apply: REMOVE Office Working hours FROM Mon I

Release Date: 26 Φεβ, 2008
Deadline: 02 Μαρ, 2008

| Sun 24/2 | Mon 25/2 | Tue 26/2 | Wed 27/2 | Thu 28/2 | Fri 29/2 | Sat 1/3 |
|---|---|---|---|---|---|---|
| 08:30 | 08:30 | 08:30 | 08:30 | 08:30 | 08:30 | 08:30 |
| 09:00 | 09:00 | 09:00 | 09:00 | 09:00 | 09:00 | 09:00 |
| 09:30 | 09:30 | 09:30 | 09:30 | 09:30 | 09:30 | 09:30 |
| 10:00 | 10:00 | 10:00 | 10:00 | 10:00 | 10:00 | 10:00 |

24/2/2008 to 1/3/2008

http://images.google.gr/imghp?tab=ci

**Figure 3.** An overall view of SELFPLANNER system. The main application window is shown at the lower left corner, now listing the active tasks. The dialog box in the bottom right is the "Edit task" window, currently shown part of the domain of the task "Prepare Slides". In the upper left corner is shown the Google Maps based application for obtaining locations. Finally, in the upper right area, Google Calendar displays the user's current plan.

SELFPLANNER supports binary constraints, in particular ordering and proximity ones, with the latter concerning minimum or maximum temporal distance between tasks. There is special treatment of these constraints when interruptible or periodic tasks are involved. As for interruptible tasks, ordering constraints between interruptible tasks apply to all pairs of their parts. A *min-distance* (*max-distance*) constraint over interruptible tasks *A* and *B* imposes that any part of *A* must be at least *min-distance* (at most *max-distance*) away from every (some) part of *B*. Higher order constraints are not supported yet.

Periodic tasks are treated similarly to interruptible tasks with one exception: If there is a binary constraint over two periodic tasks with the same period, the user has the option to consider the various iterations of the periodic tasks as individual tasks and apply the constraint repeatedly to those pairs of tasks whose periods coincide. For example, suppose an ordering constraint *A*<*B* between two weekly periodic tasks. Considering these tasks as whole means that the first iteration of *B* must start after the last iteration of *A* has been completed. On the contrary, considering each iteration of these tasks individually applies the constraint to those pairs of instances of *A* and *B* that have to be scheduled in the same week. With this interpretation, if the iterations of the two tasks are not synchronized, there might be instances of the tasks where the constraint does not apply.

Unary and binary preferences over tasks are also supported. Unary preferences concern the exact time within a task's domain when it will be scheduled. Five options are supported: none, linear-ascending, linear-descending, step-ascending and step-descending. Linear ascending/descending preferences mean that the latest/earlier a task is scheduled the better. Step ascending/descending preferences favor a task to be scheduled after/before some user-defined time point. Binary preferences involve *close* and *away* relations, meaning that two tasks are preferred to be as close to/away from each other as possible. Binary preferences over interruptible and periodic tasks are treated similarly to binary constraints.

SELFPLANNER supports incremental scheduling. Each time a new set of tasks arrives, the user may select to lock some of the old tasks at their current schedules, thus trying to schedule the new tasks at the remaining open time windows. This is a very useful feature, since it might be very annoying for the user to reschedule short-term tasks due to the arrival of new ones. However, locking the current schedule may lead to inability to schedule the new tasks or to poor schedules; in such cases the system asks the user to unlock specific tasks progressively.

SELFPLANNER also keeps track of the progress to complete each task. Current technology does not allow to monitor the user's activities accurately. On the other hand, it is very common that the user fails to accomplish scheduled tasks or parts of them. To cope with this problem, each time the user logs onto the system, she is asked about the execution status of the tasks whose scheduling time has passed. Tasks reported as not started yet are rescheduled.

Finally, in case of inability to schedule all tasks, the system gives feedback to the user about which constraints should be relaxed. Feedback is based on the best partial schedule found by the SWO search algorithm, wrt the preferences, and identifies the tasks that failed to fit into this schedule.

## Use Cases

The following are some indicative examples of tasks a user might want to insert in her calendar:

*Case 1: Attend a performance in the theater*. The performance will take place at the local theater, on Saturday evening, 21:00-23:00. This is the simplest type of task, since it is non-periodic, non-interruptible and is has a very precise time schedule and location reference. For such simple tasks, SELFPLANNER offers a *Quick Insert* functionality, in order to facilitate data entry. Furthermore, such tasks are given priority while scheduling, i.e. they are scheduled at their single possible time window before other more flexible tasks are handled.

Of course, the user has to determine the location reference of the task. This is done by defining a set of interesting locations using the accompanying Google Maps application. The user can assign any of these locations to any task; SELFPLANNER uses Google Maps to compute the time needed to travel between any pair of defined locations.

One might suggest that such simple tasks could be added directly into Google Calendar, thus bypassing SELFPLANNER. The only problem with this possibility (which is supported by SELFPLANNER) is that it is difficult to define the location reference precisely. For each task, Google Calendar provides a "Where" text field, with the possibility to view this location on Google Maps automatically. However, the mapping many times is approximately, without any possibility (at least currently) for editing. So, without having a precise location reference, SELFPLANNER does not know what temporal gap to leave between them and other adjacently scheduled tasks.

*Case 2: Buying gifts for the children*. This task uses two additional system's features: The task has neither a specific time window to schedule, nor a specific location reference. Concerning time, the user should define the duration of the task, e.g. 3 hours, a deadline, e.g. before Christmas, and the possible time slots when the task can be scheduled. In this case the time slots are the hours when the shops are open. The user can easily select these hours by applying a weekly template defining the shopping working hours to the task's domain. Note that SELFPLANNER supports applying combinations of templates, such as shops working hours but not office hours, create new templates by manually editing existing ones, as well as manually editing the domains.

Concerning the location reference, the user might be indifferent on where to buy gifts from. That is, there might be several malls in the city and the user might want to shop from the nearest one wrt to other adjacently scheduled tasks. This is achieved by defining a class of locations. A class of locations is a set of specific locations defined through the Google Maps application. For example, the user might define three different malls on Google Maps and, subsequently, define a class of locations, called Malls, that comprises the three distinct malls. Then, while defining a task, a class of locations can be used as its location reference instead of a specific location, whereas while scheduling, SELFPLANNER selects a specific location to schedule the task while trying to minimize travel times wrt adjacently scheduled tasks.

*Case 3: Writing a paper for a conference*. A well know task with specific deadline. The novelty of this task is that it is interruptible. Apart from the deadline, the user has to define its duration, e.g. 30 hours, as well as the lower and upper bounds of its parts, e.g. 2 hours and 8 hours respectively. Furthermore, the user might want not to have two 8-hours parts scheduled within the same day, so a minimum temporal distance of 12 hours between this task's parts is defined.

Again a class of locations could be used to define plausible locations to write a paper, comprising e.g. home and office. Concerning the task's domain, the user could define it by removing lunch times and sleep times from a day's hours, i.e. any hour of any day, except those devoted to sleep and eat, could be used. This is an example of a combination of two daily templates, which were retracted from the task's domain. Perhaps the user might want to remove also weekends, which could be achieved by defining a weekly template concerning weekends and removing it from the task's domain too.

Finally, suppose that the user wants to write the paper as early as possible, in order not to risk to miss the deadline. This can be achieved by assigning a linear descending preference model to this task, i.e. experienced utility decreases as time passes. Furthermore, the user might prefer not to work on the paper close to distractive tasks such as teaching classes. So, he can define a binary preference over the two tasks ('writing a paper' and 'teaching a class') stating that these two tasks should be scheduled as away to each other as possible.

*Case 4: Teaching a class*. This is a simple task like in Case 1, with the only difference that the task is weekly periodic.

However, everything is well specified, i.e. the time-window is very precise within each week (e.g. every Thursday, 14:00-17:00), whereas the same happens with the location reference (some University hall).

The procedure is similar to that described in Case 1, except that the user has to define the period (weekly) and the number of repetitions (e.g. 13 weeks). In order to define the task's domain, the user can use a weekly template, having as its only available time slots those of the class.

*Case 5: Week's shopping*. This is a combination of Case 2 and Case 4. The task is non-interruptible, admits of scheduling, and is weekly periodic. To define its domain the user has to employ a weekly template with the shopping hours. Furthermore, the user might prefer to make week's shopping as close to weekend as possible, so he can assign a linear ascending preference model to the task. Note that preference models for periodic tasks are applied separately to each instance of the task.

*Case 6: Preparing for the class*. Before each class (Case 4), the instructor wants to devote 4 hours to recap the material and possibly revise it. This task can be performed in parts, with minimum duration for each part equal to 2 hours. Furthermore, revision should be completed before the class. To capture these interrelations, the user has to define this task as a weekly periodic one, with an ordering constraint between this and the 'Teaching a class' one. Ordering constraints between periodic tasks of the same period size are applied to each pair of instances that are scheduled within the same period. So, within each week, all parts of the 'preparing a class' task should be scheduled before the 'Teaching a class' task. Note however that in this case the same effect could be achieved just by reducing the domain of the current task, since 'Teaching a class' is always scheduled in specific time-window within a week.

This section covered some of the most typical use cases one might encounter while using SELFPLANNER. What we wanted to emphasize is that, apart from solving the scheduling problem (which most of the times can be solved easily), SELFPLANNER aims at producing qualitative plans according to the user's constraints and preferences.

## Related Work

SELFPLANNER is the first system that attempts to schedule the personal tasks of a user's calendar in an automated way using constraint optimization algorithms. Furthermore, it introduces a new view of modeling this problem, including interruptible tasks, flexible periodic tasks, classes of locations, proximity constraints and preferences; such constructs do not appear in traditional calendar applications.

There are plenty of systems developed over the last fifteen years that cope with the problem of automated meeting scheduling. Some of them concentrate on specific aspects of this problem (Garrido and Sycara, 1995; Jennings and Jackson, 1995; Sen and Durfee, 1994; Sen and Durfee, 1998). More recent efforts tend to incorporate learning components or to integrate with the Semantic Web. For example, RCal (Singh, 2003) is an intelligent meeting scheduling agent that assists humans in office environments to arrange meetings. RCal agents negotiate with each other on the behalf of their users and agree on a common meeting time that is acceptable to all the users and abides by all the constraints set by all the attendees. RCal supports parsing and reasoning about semantically annotated schedules over the web, such as conference programs or recurring appointments (Payne, Singh and Sycara, 2002).

CMRadar (Modi et. al, 2004) is an end-to-end agent for automated calendar management that automates meeting scheduling by providing a spectrum of capabilities ranging from natural language processing of incoming scheduling-related e-mails, to negotiate with other users or making autonomous scheduling decisions.

PTIME (Berry et. al., 2006) is an ongoing effort being developed under the CALO project (Myers, 2006), that aims at facilitating meeting scheduling. The innovation of PTIME lies at its capability to learn the user's preferences thus adapting its future behavior, whereas it emphasizes in adopting natural language for interfacing with the user. A more recent effort within the same project concerns Towel (Conley and Carpenter, 2007), an initial attempt towards an intelligent to-do list. Towel allows the user to organize to-dos (group, tag, check etc) as well as delegate them to other users or agents. Although to-dos can be seen as tasks, Towel emphasizes on to-dos manipulation rather than in solving the scheduling problem associated with actually performing these to-dos. Furthermore, it does not support all the advanced modeling features of SELFPLANNER.

## Conclusions and Future Work

SELFPLANNER is currently a research prototype. Its evolvement is based on users' suggestions from actual use. The system is publicly available through a moderated web-based registration procedure[2]. Currently, its users sum to a few decades, most of them from our home institute. However, those who use it systematically for their calendar needs are less than 10, including the authors.

As it is clear from the previous section, most of the effort to add intelligence to calendar applications concentrates on automating meeting scheduling. We could imagine two explanations for this: First, people think that meeting scheduling is the most difficult and time consuming part of organizing a user's time. Although this might be true, meetings constitute a small part of our daily

[2] http://selfplanner.uom.gr. An earlier version of the system has been presented at the demo session of ICAPS-07.

activities, whereas poor organization of the remaining tasks may result in significant waste of time. Indeed, even scheduling together tasks with the same location reference, in order to avoid pointless moves, would be of great value for many users.

On the other hand, we believe that the main reason why intelligent calendar systems do not focus yet on automated scheduling of personal tasks is due to an underlying consensus that it would be very difficult for a user to accept a machine-generated daily plan of activities, for psychological reasons among others. Our experience from using the system is that this is not absolutely true: First of all, many of a user's tasks are very constrained, as for example giving a lecture or getting the children to school, so there is no need of scheduling for them. For the rest of the tasks, SELFPLANNER gives the user so many options to constrain the schedule and express her preferences, that it is very unlikely to get an unacceptable plan. In any case, our experience has shown that through the actual interaction with the system people learn personal policies of how to use it and get the most from it.

As for the psychological objections, we can witness our evidence: SELFPLANNER has emerged as a way to fulfill personal organization needs and continues to evolve based on its users' feedback. The initial motivation was to develop an intelligent calendar system able to schedule together tasks with the same location reference. However, through the actual use of the first prototype, several other needs came up, with many of them being already implemented as described in this paper. In any case, we have to admit that the system's current user base does not consist of average users but from computer specialists. So, further investigation is required if we want this system to be adopted by the general public.

As for the future, there are several directions into which SELFPLANNER can be extended. Our experience suggests that the key to success and broader adoption is the easiness of use. We receive constantly suggestions by the users for new functions that facilitate data entry. Our immediate plans include a mobile interface, as well as manually editing of the current plan (currently the current plan is read only over Google). Integration of meeting scheduling capabilities is also under consideration.

However, the most challenging extension concerns its transformation to a planning system. This next generation system will possess semantic knowledge about its user's status, a rich ontology with actions having preconditions and effects and a set of goals to be achieved. The system will help the user to insert tasks into her calendar in order to achieve goals or subgoals. For example, the user might set a goal for attending a conference, which could generate a sequence of actions being inserted into her calendar, including preparing the slides, bookings for the trip and travelling. Classical planning algorithms for causal reasoning can be used to solve the planning problem, which can be done in a mixed-initiative manner.

## References

Berry, P., Conley, K., Gervasio, M., Peintner, B., Uribe T. and Yorke-Smith, N. Deploying a Personalized Time Management Agent, 5th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-06) Industrial Track, Hakodate, Japan, pp. 1564-1571, May 2006.

Conley, K., and Carpenter. Towel: Towards an Intelligent To-Do List, AAAI Spring Symposium on Interaction Challenges for Artificial Assistants, Stanford, CA, March 2007.

Garrido, L. and Sycara, K. Multi-agent meeting scheduling: Preliminary experimental results, 1st International Conference on Multi-Agent Systems (ICMAS), 1995.

Jennings N.R. and Jackson, A.J. Agent based meeting scheduling: A design and implementation, *IEE Electronic Letters*, 31(5):350-352, 1995.

Joslin, D.E., and Clements, D.P. "Squeaky Wheel" Optimization, *Journal of Artificial Intelligence Research*, vol. 10 (1999), 375-397.

Modi, P.J., Veloso, M., Smith, S.F. and Oh, J. CMRadar: A Personal Assisstant Agent for Calendar Management, Workshop on Agent Oriented Information Systems (AOIS), New York, 2004.

Myers, K. Building an Intelligent Personal Assistant, AAAI Invited Talk, 2006.

Payne, T.R., Singh, R. and Sycara, K. Calendar Agents on the Semantic Web, IEEE Intelligent Systems, vol. 17(3), pp84-86, May/June 2002.

Refanidis, I. Managing Personal Tasks with Time Constraints and Preferences, 17th International Conference on Automated Planning and Scheduling Systems (ICAPS-2007), Providence, Rhode Island, 2007.

Refanidis, I., McCluskey, T.L. and Dimopoulos, Y. Planning Services for Individuals: A New Challenge for the Planning Community, Workshop on Connecting Planning Theory with Practice, Whistler, British Columbia, Canada, 2004.

Sen, S. and Durfee, E.H. A formal study of distributed meeting scheduling, *Group Decision and Negotiation*, vol.7, pp. 265-289, 1998.

Sen, S. and Durfee, E.H. On the design of an adaptive meeting scheduler, 10th International Conference on Artificial Intelligence for Applications, pp. 40-46, 1994.

Singh, R. RCal: An Autonomous Agent for Intelligent Distributed Meeting Scheduling, master's thesis, tech. report CMU-RI-TR-03-46, Robotics Institute, Carnegie Mellon University, December, 2003.

Van Hentenryck, P., Saraswat, V. and Deville, Y. Design, implementation and evaluation of the constraint language cc(fd), *J.of Logic Programming*, 37 (1998), pp. 139-164.