

Time Management in the Intelligent Assistant

Wayne Wobcke

Intelligent Systems Research Group
BT Labs, Adastral Park, Martlesham Heath
Ipswich IP5 3RE, United Kingdom
wobckew@info.bt.co.uk

Abstract. The Intelligent Assistant (IA) is a system of software agents for helping the user with communication, information and time management. In this chapter, we discuss in detail issues related to time management. There are two distinct types of time management relevant to the IA: that concerning the user's management of his or her own time, and that concerning the coordination of actions performed by the various specialist assistants in the system (which affects the overall effectiveness of the system from the user's point of view). To aid the user in managing his or her own time, the IA includes a *Diary Assistant* which acts as a scheduler of tasks with the aim of satisfying the user's combined preferences for start times, durations and deadlines. The Diary Assistant offers ease of use by allowing preferences for a task to be specified using natural language terms such as *morning*, *afternoon*, *early morning* and *around 11:00*, which are interpreted by the system using fuzzy functions. To manage the system's time, the IA has a special *Coordinator* for regulating the communication from the system to the user and for planning system tasks. The Coordinator is the only component of the IA capable of scheduling the future actions of the assistants, and incorporates a novel agent architecture based on ideas from reactive scheduling called IRSA (*Intelligent Reactive Scheduling Architecture*). The Coordinator constructs and maintains the system's schedule of tasks using information about the user's schedule obtained from the Diary Assistant.

1 Introduction

The Intelligent Assistant (IA) is a collection of software agents for helping the user in aspects of communication, information and time management. The IA includes a number of specialist assistants: the Telephone Assistant for filtering telephone calls, the E-mail Assistant for prioritizing messages, the Web Assistant for web searches, the YPA for classified directory enquiries, and the Diary Assistant for calendar management. Each of these assistants has its own user interface and may need to repeatedly communicate with the user.

The work described in this chapter addresses issues of time management in the Intelligent Assistant. Time management in the IA takes two forms: first, the *Diary Assistant* helps the user manage his or her own time, and second, the IA has a *Coordinator* that helps to manage the system's time, both in controlling (to some degree) communication between the assistants and the user, and in

scheduling the system's own future actions. Regulation of communication to the user is needed to prevent potential confusion and communication overload arising from a number of assistants attempting to grab the user's attention at the same time; the scheduling of the system's future actions concerns the efficient use of system resources (e.g. web search) and so relates to the timely presentation of information from the assistants. The two agents are thus very different, although they perform the same basic function, that of task scheduling.

The purpose of the Diary Assistant, being a tool under the user's control, is to construct schedules that satisfy the user's stated preferences for task start times, durations and deadlines. The system is personalized in taking into account user preferences when scheduling tasks. It makes use of fuzzy logic in allowing the user to specify preferences for the start time, duration and deadline of a task using simple natural language expressions such as *morning*, *afternoon*, *early morning*, *around 1:00*, etc. The assistant uses two scheduling algorithms: a *global* search is used when scheduling a set of tasks, while a *local* search is used when adding a single task to an existing day schedule.

The purpose of the Coordinator is to maintain information about the user's state and activities and to schedule future actions of the system: it is distinguished as the only agent in the IA with a scheduling capability. The Coordinator is an instantiation of a new agent architecture based on ideas from reactive scheduling. IRSA (*Intelligent Reactive Scheduling Architecture*) explicitly addresses questions relating to intentions in time, and focuses on the issues of intention maintenance and revision. IRSA extends the PRS system of Georgeff and Lansky [1] to include a model of time, allowing the interpreter to schedule its own future actions and then to execute those actions at the scheduled times.

The organization of this chapter is as follows. The main body of the material is contained in two sections, section 2 describing the Diary Assistant and section 3 describing the Coordinator. In section 2, after some motivating discussion, we present the interface to the Diary Assistant, the fuzzy evaluation function used by the schedulers, and the global and local scheduling algorithms. In section 3, we give an overview of the abstract IRSA architecture, provide details of the implementation of the Coordinator, and present a scenario illustrating the integration of the assistants in the IA. We conclude in section 4.

2 Diary Assistant

Present diary systems are of limited assistance: they act mainly as electronic versions of paper diaries that can be used as a record of appointments or for reminders of tasks to be done. In this section, we describe a *Diary Assistant*¹ which helps its users schedule personal tasks. The system is personalized in taking into account user preferences when scheduling tasks. It makes use of fuzzy logic, along the line of Zadeh [2], in allowing the user to specify preferences for the start time, duration and deadline of a task using simple natural language

¹ An initial version of the Diary Assistant was implemented by Simon Case.

expressions such as *morning*, *afternoon*, *early morning*, *around 1:00*, etc. Each preference is interpreted by the system using a predefined fuzzy function. The Diary Assistant uses two schedulers: a *global* scheduler that assigns timeslots to a set of tasks so as to maximize the overall satisfaction of preferences, and a *local* scheduler that allocates a timeslot to a single task so as to minimize disruption to an existing day schedule. Both algorithms make use of an evaluation function that computes the degree to which a partial schedule satisfies a set of user preferences. The local scheduler uses an iterative improvement strategy based on the heuristic of minimizing changes to task sequences, while the global scheduler uses a standard greedy search algorithm driven by the evaluation function.

The general motivation for our work on the Diary Assistant is the personal assistant metaphor promoted by Maes [3] as characterizing a class of software agents that learn from the user as he or she performs a repetitive task over a period of time. The idea was that a system could adapt its behaviour to the user's habits by "looking over the shoulder" of the user, and hence could improve its performance with only minimal intervention. Some early systems were also known as *interface agents* because they typically provided a single point of contact between a user and a single software system such as a mail handler or calendar system. The calendar system described in Maes and Kozierek [4] used a combination of memory-based reasoning and reinforcement learning to adapt to the user's habits in a very specific scenario, here deciding whether to accept, decline or request renegotiation of a proposed meeting time. Thus the system provided only minimal help to the user.

Another related system is the CAP system described by Dent *et al.* [5]. This work derives from the "learning apprentice" metaphor, and uses a standard machine learning algorithm to learn the preferences of a single user for scheduling various types of meetings. In contrast to this system, our work focuses on task scheduling based on expressed user preferences, whilst learning those preferences is something which we have so far not investigated.

Our approach to specifying user preferences draws on what Zadeh [2] calls 'computing with words'. In Zadeh's proposal, the vagueness of natural language terms is represented using fuzzy functions (functions whose range is the interval $[0, 1]$), the idea being that the meaning of a word like *tall* is captured by associating a degree of satisfaction of the predicate for each actual height. So, for example, 1.9 metres may be tall to degree 1, while 1.8 metres may be tall to degree 0.9, etc. Our idea is that preferences expressed in natural language terms such as *early morning* also denote fuzzy functions; thus preferences can also be satisfied to greater or lesser degree. Fuzzy logic provides a simple way of combining preferences: in fuzzy logic, the degree of satisfaction of two preferences is the minimum of the degrees to which each preference is satisfied (although more complicated mechanisms are also possible).

The present work thus closely borrows from that on fuzzy scheduling, which has, however, been concerned mainly with job-shop scheduling and has not so far been combined with 'computing with words': perhaps the earliest work in this area is Prade [6]. One theoretically motivated approach is described by Dubois,

Fargier and Prade [7], who present a constraint analysis algorithm adapted to fuzzy temporal intervals using the approach of Dubois and Prade [8]; Kerr and Walker [9] is an early application of fuzzy logic to job-shop scheduling, and Slany [10] discusses alternative methods of combining multiple constraints. These systems are all designed for capturing and reasoning about the relative importance of constraints (rather than preferences) associated with release times and due dates of jobs: more precisely, that some “soft” constraints are free to be relaxed (to varying degrees), while other “hard” constraints are absolute. In addition, Türkşen [11] gives an overview of three other ways that fuzzy methods can be applied in the design of scheduling systems.

The Diary Assistant makes use of a “local” scheduler for single user tasks. Our use of iterative improvement techniques for incremental scheduling is similar to its use in BT’s Dynamic Scheduler, Lesaint *et al.* [12], which applies the technique to a constrained version of the Vehicle Routing Problem. The method itself borrows more generally from work on reactive scheduling, e.g. Smith [13, 14], which is primarily in the domain of job-shop scheduling. He proposes a general scheduling architecture, OPIS, based on a blackboard control structure; in addition to incremental scheduling, OPIS provides heuristics for schedule repair when a schedule becomes infeasible. However, both in Dynamic Scheduler and in OPIS, the heuristics for schedule modification are necessarily specific to the problem domain.

In the remainder of this section, we describe the interface to the Diary Assistant, the fuzzy evaluation function used by the system, and the global and local scheduling algorithms.

2.1 Interface

The interface to the Diary Assistant is designed to look like a standard paper diary, with each day divided into half hour slots from 9:00 a.m to 5:00 p.m. Each task has the following features: (i) preferred start time, (ii) preferred start day, (iii) duration (based on 30 minute slots), (iv) description, (v) deadline time, (vi) deadline day, and (vii) interruptible (a flag indicating whether the user is interruptible during the task). The description may be selected from a predefined hierarchy of task descriptions provided for convenience, such as *Meeting/Research*. The diary window displays three days, and the user can move the window one day into the past or the future by clicking on arrow buttons at the bottom of the screen. The main interface window is shown in Figure 1: the duration of each task is given in parentheses beside its description, in the form *hours:minutes*.

The system facilitates the easy addition, deletion and movement of tasks from one day to another. When a task is moved to a new day, the scheduler uses the original preferred start time, so the timeslot allocated to the task on the new day may differ from previously. For example, suppose an *early morning* task is scheduled at 9:30 a.m. on one day; if this task is moved to a completely free day, it will be scheduled at 9:00 a.m., the time at which the *early-morning* function is maximized (see Figure 2 below). There is also a “To-Do” list containing tasks as yet unscheduled.



Fig. 1. Diary Assistant Interface

By clicking twice on a task, the user indicates that the task is to remain committed to its current timeslot, or, if the task is in the To-Do list, that it should remain unscheduled. The global scheduler, at the command of the user, reschedules all tasks on the To-Do list not indicated as remaining unscheduled. The local scheduler is used when a new task is added, when a task is moved from one day to another, and when the time, day, duration or deadline of a scheduled task is changed by editing.

2.2 Evaluation Function

Users can specify preferences for the start time, duration and deadline of a task using terms that are interpreted as fuzzy functions. At present, the following fuzzy terms are supported: for time preferences *morning*, *afternoon*, *early morning*, *late morning*, *early afternoon*, *late afternoon*, *around t* and *about t* where t is a precise time, and when using the global scheduler, *early week*, *mid week* and *late this week*; and for durations *around d* and *about d* where d is a precise duration, e.g. *around 1:00* (meaning a duration of around 1 hour). The function denoted by each of these predicates is predefined: it is our eventual aim to have these adaptively learnt by the system. As an illustration, fuzzy functions for interpreting various predicates for day periods are shown in Figure 2. As can be seen, the functions used are typically triangular and trapezoidal fuzzy functions.

The fuzzy function for the degree to which a time length l meets a duration of *around d* or *about d* is as follows.

$$\mu_t(l) = \begin{cases} 1 & \text{if } l = d \\ 0.3 & \text{if } l = d \pm 30 \text{ minutes} \\ 0 & \text{otherwise} \end{cases}$$

Because of the low value either side of the maximum (0.3), this function reflects a strong preference for changing a task's start time rather than shortening its duration (see below).

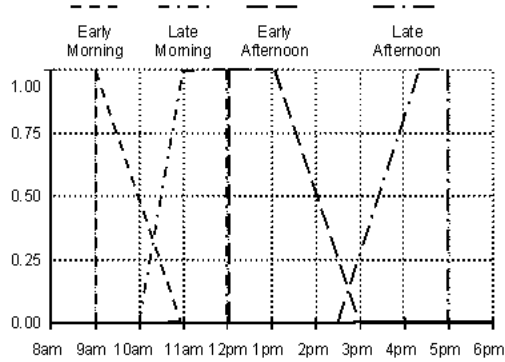


Fig. 2. Sample Fuzzy Functions

Finally, the degree to which a time s meets a deadline t is defined as follows, using the fuzzy function $\mu_{(-\infty, t]}$, assuming that μ_t is defined, see Dubois and Prade [8].

$$\mu_{(-\infty, t]}(s) = \sup_{s \leq x} \mu_t(x) = \begin{cases} 1 & \text{if } s \leq t^* \\ \mu_t(s) & \text{otherwise} \end{cases}$$

where t^* is the latest timepoint x for which $\mu_t(x) = 1$, assuming that $\mu_t(x)$ is decreasing for $x \geq t^*$.

To combine these functions with a scheduling procedure, we need a way of computing the overall degree of satisfaction for several preferences that pertain to the same task, and a way of extending the evaluation function to cover several tasks in a partial task allocation. The general problem is one of aggregating preferences. Let P be a set of preference values which the user provides for a series of features. Here the features are the start time, duration and deadline of a diary task. Let T be a set of tasks and α be an allocation of timeslots (not necessarily of half hour length) to each task in T , so that $\alpha(t)$ is a timeslot for each $t \in T$. For each preference value $p \in P$, there is presumed to be a function μ_p giving the degree to which the timeslot s satisfies the preference p . It is natural to define the function μ_P representing the degree to which a timeslot s satisfies a set of preferences P as the minimum of the degrees to which s satisfies the individual preferences in P , since satisfying each of a set of preferences is analogous to satisfying the conjunction of those preferences. That is, we have assumed (along with both classical and fuzzy logic) that the preferences are independent. Thus we define:

$$\mu_P(s) = \min_{p \in P} \mu_p(s)$$

Note that if a timeslot violates any one of the preferences in P , the value of this function is 0.

Now the scheduling problem is to find an allocation α to a set of tasks that maximizes overall preference satisfaction. The scheduler typically operates with

partial solutions (i.e. task allocations to a set of tasks that satisfy all preferences on those tasks). So that the search progresses towards a solution, we need a function μ for evaluating allocations to varying task sets T that is monotonic in T and α , i.e. if $T_1 \subseteq T_2$ and $\alpha_1 \subseteq \alpha_2$ then $\mu(T_1, \alpha_1) \leq \mu(T_2, \alpha_2)$. The function we choose is the summation function, although no doubt many other functions would work just as well. Thus we define (for a fixed set of preferences P):

$$\mu(T, \alpha) = \sum_{t \in T} \mu_P(\alpha(t))$$

In summary, equation (1) is the function we use for evaluating partial task assignments, where T is a set of tasks and α is an allocation of timeslots to the tasks in T that satisfy all preferences on those tasks, and $\mu_p(s)$ is the degree to which a timeslot s satisfies a preference value p .

$$\mu(T, \alpha) = \sum_{t \in T} \min_{p \in P} \mu_p(\alpha(t)) \quad (1)$$

2.3 Global Scheduler

The Diary Assistant uses two schedulers: a *global* scheduler that allocates timeslots a set of tasks so as to maximize overall preference satisfaction, and a *local* scheduler that makes minimal adjustments to an existing schedule when adding (or editing) a single new task. The two scheduling algorithms are implemented in Fril, Baldwin, Martin and Pilsworth [15], a logic programming language incorporating features of fuzzy inference, although the fuzzy aspects of Fril are not used to implement the evaluation function in this application.

We first describe the global scheduling algorithm. The system starts with a partial schedule for a set of tasks which the user has designated as fixed. The object is to find an assignment α to a set of tasks T such that $\mu(T, \alpha)$ is maximized. The problem, as with other scheduling problems, can be construed as a search problem. However, in contrast to other scheduling problems such as job-shop scheduling, the problem is relatively unconstrained, so there are typically many solutions that need evaluation. Hence the global scheduler uses a standard depth first search algorithm with backtracking, with heuristic guidance to improve the efficiency of the search.

The search uses heuristics in two ways: for defining the order in which tasks are considered and the order in which allocations to tasks are explored. Tasks are ordered from most constrained to least constrained, where one task is more constrained than another if there are fewer possible timeslots that satisfy the user's preference for that task. Note that the degree of satisfaction is not taken into account, just the number of possibilities: for example, an *early morning* task is more constrained than an *afternoon* task. Ties are broken arbitrarily. When considering a single task in the context of a partial schedule (an allocation of timeslots to the more constrained tasks), the possible timeslots are explored in order of preference satisfaction. This makes it highly likely, although there is no

guarantee, that the first solution found by the scheduler (assuming one is found) maximizes the evaluation function μ defined in equation (1): for efficiency, the first solution found is returned, even though this may not be optimal.

A sketch of the algorithm is given in Figure 3. It is assumed that a task allocation is represented as a set of pairs $\langle t : s \rangle$ where t is a task and s a timeslot. The first call to **Schedule** has T initialized to the set of all tasks ordered from most constrained to least constrained, and A to the empty set. The set S of free timeslots in an allocation that satisfy a preference p is ordered in decreasing degree of satisfaction of p .

```

Schedule(TaskSet T, Allocation A):
{
  if (T = {})
    return A;    % success
  else
    t = first(T); T = rest(T);
    p = user preference for t;
    S = free timeslots in A satisfying p;
    if (S = {})
      return null;    % backtrack
    else
      repeat
        s = first(S); S = rest(S);
        A' = Schedule(T, A  $\cup$  { $\langle t : s \rangle$ })
      until (A'  $\neq$  null) or (S = {});
      return A';
}

```

Fig. 3. Global Scheduling Algorithm

2.4 Local Scheduler

The idea behind the use of local scheduling in the Diary Assistant is to minimize disruption to an existing schedule when a *single* task is added or modified. The assumption here is that it is desirable to minimize as far as possible any confusion caused by moving diary entries, so tasks are not moved unless this is necessary for the satisfaction of preferences. Note that in the current system the day of a new task is supplied by the user, so the search is limited to a single day, and hence is much more constrained than global search.

We use a “hill-climbing” iterative improvement algorithm to implement local search. The algorithm presupposes a mechanism to compute the “neighbouring” solutions to a given solution state, which are typically found by modifying the solution by applying some simple heuristics (we use the term ‘solution’ here to

indicate a task allocation that satisfies all the preferences). The heuristic we employ for generating neighbouring states is to consider each legitimate state computed by moving a sequence of tasks of up to length three either forwards or backwards by half an hour. The aim is to minimize changes to *task sequences* in the user's schedules. This is only one heuristic that we feel is intuitively reasonable. We prefer it to, for example, swapping tasks, either adjacent or nonadjacent. Of course the iterative improvement algorithm can operate with any heuristic.

A further issue is the construction of the initial solution. Here we also allow task sequences of up to length three to be moved forwards or backwards by half an hour, and tasks of imprecise preferred duration to be compressed (possibly in conjunction with moving a preceding or succeeding task half an hour forwards or backwards). This is because sometimes there is no gap in the schedule of sufficient length in which to place the new task; moving or shortening tasks sometimes creates such a gap.

A sketch of the modification algorithm is given in Figure 4. It is assumed that the initial solution A is found by the heuristic means of allowing movement and compression of tasks as described above. The score is just the value computed by the μ function, equation (1), for complete solutions.

```

Schedule(Allocation A):
{
  N = neighbouring solution states of A;
  if (N = {})
    return A;
  else
    A' = best state in N;
    if (score(A') ≤ score(A))
      return A;
    else
      return Schedule(A');
}

```

Fig. 4. Local Scheduling Algorithm

The typical case where the use of local search improves the solution is when there are a number of relatively unconstrained tasks that require scheduling. For example, suppose there are two 1 hour afternoon tasks, which have been scheduled for 2:30 and 3:30. A new 1 hour afternoon task is to be added. The system initially assigns the new task to 1:30, then discovers that by moving all tasks backward by half an hour (i.e. the new task to 2:00, the 2:30 task to 3:00, and the 3:30 task to 4:00), the user's preferences are better satisfied.

3 Coordination in the Intelligent Assistant

The problem addressed in this section is the coordination of the various assistants in the Intelligent Assistant. Coordination is necessary first to avoid overloading the user with information from many assistants at the same time: this type of coordination essentially involves managing the communication from the assistants to the user. A more interesting aspect of coordination involves proactive behaviour: this refers to the system itself performing tasks on behalf of the user that require the action of more than one assistant (comprising simple plans of action). A subsidiary aspect of this type of coordination is the performance of actions at appropriate times. As a simple example, a reminder of a meeting should be given at a time close to the meeting and at a time when the user is likely to be available to receive the reminder. The reminder also has a deadline (the start time of the meeting) after which attempting the action is useless. Moreover, if the time of the meeting changes, the time of the reminder must also change. Thus temporal reasoning is a central part of coordination in this sense.

In this section, we describe the method of coordination we have developed to solve these problems. We have implemented a special *Coordinator*² which is capable of scheduling and subsequently executing its own future actions: the Coordinator is distinguished within the IA as the only assistant with a scheduling capability. To support this facility, the Coordinator maintains a temporal database of the user's planned activities (obtained from the Diary Assistant), and uses this both to manage the system's interactions with the user and to schedule system actions related to the user's tasks. The Coordinator maintains this model as entries in the diary change, and reschedules its actions if necessary. An important point is that, due to the IA's messaging architecture, the Coordinator should not be regarded as a centralized controller: although the Coordinator can request other assistants to perform actions, the other assistants are autonomous to the extent that (i) they may not perform those actions, and (ii) they operate continuously, and hence for much of the time are operating under the direction of the user, rather than the Coordinator.

The Coordinator is based on a novel agent architecture that combines ideas from agent technology and reactive scheduling. As the name indicates, IRSA (*Intelligent Reactive Scheduling Architecture*), has much in common with previous architectures that follow Bratman's theory of intention, Bratman [16], particularly IRMA, Bratman, Israel and Pollack [17], and PRS, Georgeff and Lansky [1], and exemplifies a cautious approach to the design of agent systems. In essence, IRSA extends the PRS system with a model of time, enabling the system to schedule and subsequently execute its own future actions. But where the inspiration for PRS seems to be the multi-processing operating system with a view of a plan as an interruptible program, i.e. sequence of actions, the IRSA system views a plan as a set of actions each scheduled for execution at a particular point in the future. The architecture thus borrows from reactive scheduling, e.g. Smith [13, 14], in its focus on intention maintenance and revision in response to

² An initial version of the Coordinator was implemented by Arash Sichanie.

changes in the world, in contrast to the PRS use of interrupt mechanisms to recover from plan execution failures. In support of these functions, the IRSA world model uses a temporal database to represent the future, while the PRS database typically contains information only about the current state of the world.

The Coordinator must maintain the timeliness of its scheduled actions. This sometimes means that tasks need to be rescheduled when the user task (goal) time is changed in the diary. For example, when the user changes the time of a meeting, the Coordinator must reschedule the reminder to the meeting. In the present implementation, this is done by simply deleting the goal and all its associated tasks (upon notification of the change from the diary), creating a new goal corresponding to the new time and then scheduling new tasks in response to the new goal. Another need for rescheduling occurs when the time comes for an action to be executed which involves notifying the user (such as issuing a reminder) but the user is not reachable for some reason (the Coordinator's model of the user's tasks is not necessarily complete). In this case, the notification is rescheduled for the next time the user is known to be reachable according to the presently available information (unless this time is past the task's deadline, in which case the goal is dropped).

The sense of "coordination" used for the IA is somewhat different from other senses used in the literature, which tend to involve negotiation or joint commitments between multiple agents using a protocol such as the contract net, c.f. Smith [18], Jennings [19]. It is closer to the sense of coordination in Sycara and Zeng [20], who propose the TCA (Task Control Architecture), in which multiple task agents are connected to multiple interface agents and multiple information agents. This system has been applied to a number of problems where one task agent forms a plan that involves the action of other agents. However, in their applications, only one agent, an interface agent, communicates with any one user. Our work on the IRSA architecture is related to work on integrating planning and reacting in the AI planning literature, and in its use of scheduling to partial global planning, Durfee and Lesser [21], and generalized partial global planning, Decker and Lesser [22], which is also based on agents that employ scheduling algorithms. A major focus of that work is the coordination of a number of scheduling agents that negotiate to form a common schedule, although the agents do not execute their own schedules. Also many of the applications are scheduling-type examples in which resource optimization is the overall aim. At present, the IRSA agent implemented as the Coordinator of the IA does not coordinate its schedules with other similar agents, but only coordinates (in a more loose sense) the actions of itself and other autonomous nonscheduling agents. We sometimes refer to our notion of coordination as orchestration, as the Coordinator plays a role analogous to the conductor of an orchestra.

The remainder of this section is organized as follows. In section 3.1, we give an overview of the abstract IRSA architecture. Details of the interpreter and the implementation are provided in section 3.2. In section 3.3, we illustrate the domain specific aspects of the architecture as instantiated as the Coordinator of the Intelligent Assistant through the use of a simple scenario.

3.1 IRSA Architecture

In this section, we describe our agent architecture which is called IRSA (*Intelligent Reactive Scheduling Architecture*). IRSA is both an abstract architecture, which is described in this section, and a concrete realization of this architecture, described in section 3.2. So more precisely, it is the implementation of the architecture that meets the needs of schedule maintenance and revision, and the claim is that it does this in virtue of implementing the abstract architecture. Moreover, the abstract architecture leaves open the particular methods through which schedule maintenance and revision are implemented, as we believe that these are largely domain dependent.

In common with many agent architectures that aim to integrate planning with reactivity, e.g. Firby [23], Ferguson [24], Gat [25], Lyons and Hendriks [26], Müller and Pischel [27], Bonasso *et al.* [28], and especially Sloman [29], IRSA is a layered architecture, consisting of three layers as shown in Figure 5: a reactive layer, a deliberative layer and a “meta-reasoning” layer.

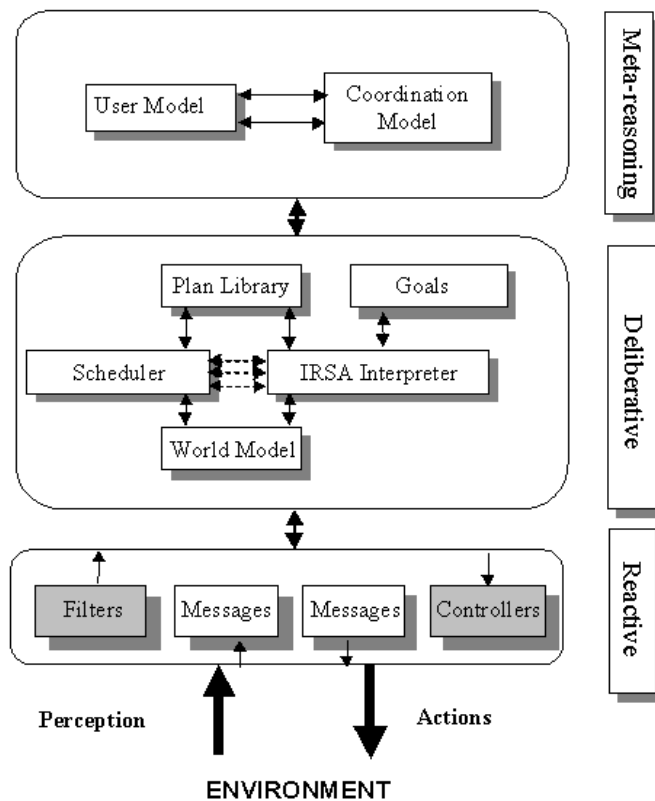


Fig. 5. IRSA Agent Architecture

The reactive layer is responsible for dealing with inputs from the environment that require an immediate response (by which we mean that there is insufficient time for any deliberation to influence the action taken). In the IA application, this “environment” consists only of the user and the other assistants, so the possible inputs are messages from other assistants, information about the user’s state obtained from the multi-modal interface, and commands from the user. In general, this layer would also include filters and controllers.

The deliberative layer contains the IRSA architecture proper, which is described more fully in section 3.2. It is responsible for accepting goals (which in the case of the IA correspond to user tasks), retrieving plans appropriate to those goals, scheduling actions in the plan(s), maintaining the plans as time progresses, and executing each scheduled action at the allocated time. Also included at this layer is a “world model” used when scheduling actions, and the system is responsible for maintaining the accuracy of this model: for example, in the IA when a user changes the time of a task using their diary, the Coordinator must both update the world model and reschedule the actions scheduled in response to that user task. The depiction of this layer in Figure 5 is meant to emphasize the relationship between the IRSA architecture and the PRS system of Georgeff and Lansky [1] and does not accurately reflect all the modules in the implementation (see section 3.2 below for these details).

The meta-reasoning layer is currently the least developed of the three layers in the present implementation. It includes a user model, which in the case of the Coordinator consists of the user’s preferred times for performing various tasks. This enables the system to schedule actions in response to various system internal goals (such as issuing reminders to read low priority e-mails), which repeatedly arise and which we therefore call *recurrent* goals. The times are expressed using fuzzy terms, so an example fact in such a model is ‘likes to read daily e-mail in late afternoon’. At a particular time each day (presently 11:00 p.m.), the Coordinator sets up internal goals corresponding to such tasks, e.g. *read daily e-mails*. However, the action corresponding to this goal is not to schedule a reminder to read daily e-mail, but rather to schedule a time at which the reminder will be scheduled. When determining the time for later scheduling, the user model is consulted so that (in this case), the scheduling action is performed some time before “late afternoon”. Then at that time, the actual reminder is scheduled, and eventually, the reminder issued. The reason behind this “meta-scheduling” is that when the initial recurrent goal is generated (at 11:00 p.m.), the Coordinator is not assumed to have accurate information about the user’s day, so scheduling the reminder then would be useless (or else the time of the reminder would require rescheduling in the event of a conflict). To avoid this, a scheduling event is scheduled for a time when the system can be expected to have more complete information.

The meta-reasoning layer also contains provision for a “coordination model”, here referring to coordination of the activities of different IA users. The type of application we have in mind is multi-agent meeting scheduling, an obvious extension to the Diary Assistant requiring the coordination of multiple users.

3.2 Implementation

In this section, we focus on the deliberative layer of the IRSA architecture, which implements the planning/action interpreter of the system. As mentioned above, the IRSA interpreter is primarily an extension of the BDI interpreter of PRS, Rao and Georgeff [30], that enables intentions to be scheduled at specific points in time. The implementation also represents an advance over PRS (at least of the UM-PRS implementation, Lee *et al.* [31], in its use of Java threads for implementing concurrent processes. In fact, the goal reduction planner is only one of three threads forming the implementation of the deliberative layer. A second thread is used for executing the actions scheduled by the interpreter, and a third thread is responsible for updating the world model in response to changes in the environment (in this case, a change in the user's state).

The basic IRSA interpreter implements a goal reduction planner as shown in Figure 6. This thread takes each of the system's goals, retrieves a plan for the goal from a pre-specified plan library, and calls the scheduler to specify execution times for system tasks for each action in the plan. Goals mainly correspond to user tasks, so each goal comes with a deadline time (set to the start time of the user's task). Each plan in the plan library consists of a set of action templates for achieving a particular goal. Before a system task can be scheduled, the action template from which it derives must be instantiated to form an executable task using the parameters supplied with the specific goal. There are currently two types of scheduling for actions: either 'as soon as possible' or 'as late as possible before the deadline', the former typically being used for search tasks or notifications, and the latter for reminders (in this case, the scheduler determines a time which is 5 minutes before the user becomes unreachable from then until the action's deadline). System-generated "recurrent" internal goals are treated by the interpreter in exactly the same manner.

```
Interpreter:
{
  // basic goal reduction planner
  Goal = GoalList.getGoal();
  if (Goal ≠ null)
  {
    Plan = PlanLibrary.fetchPlan(Goal);
    if (Plan ≠ null)
      Scheduler.schedule(Goal, Plan);
    Goal.setStatus(REduced);
  }
}
```

Fig. 6. IRSA Interpreter

The IRSA execution module is illustrated in Figure 7. This thread repeatedly, at an interval appropriate to the application, checks whether there is a task that needs executing, then executes it. For the IA, the thread runs every 15 seconds, which is generally enough time to execute all tasks scheduled for a particular time. Note that the executor does not assume that there is only a single task scheduled for a particular time, but note also that the tasks are executed sequentially, not concurrently. The world monitor for the Intelligent Assistant is necessarily domain dependent, and is described in the next section.

```
Executor:
{
    // assumes now is the current time
    Task = TaskList.getTask(now);
    while (Task ≠ null)
    {
        executeTask(Task);
        Task.setStatus(COMPLETED);
        Task = TaskList.getTask(now);
    }
}
```

Fig. 7. IRSA Execution Module

The interpreter is also capable of simple “rescheduling” in response to changes in the world model. For example, when the user changes the time of a task in the diary, the Coordinator must reschedule the actions associated with the task. In the present implementation, this is done by simply deleting the goal and all its associated tasks, creating a new goal corresponding to the new time and then scheduling new system tasks in response to the new goal. This simple mechanism suffices for the relatively small plans currently used in this application.

3.3 Integration

In this section, we illustrate the integration and coordination of the assistants in the Intelligent Assistant. The IA is designed as a multi-agent system consisting of various specialist “assistants” in the areas of time, information and communication management. For communication management, the IA includes specialist Telephone and E-mail assistants, used respectively for filtering telephone calls and prioritizing e-mail messages. For information management, the IA includes a Web Assistant (for web search) and the YPA (for Yellow Pages^{®3} lookup).

³ Yellow Pages[®] is a registered trade mark of British Telecommunications plc in the United Kingdom.

Communication between agents in the IA is implemented using the open messaging architecture of Zeus, Nwana, Ndumu and Lee [32]. The interaction between the various agents is shown in Figure 8, which shows the types of messages or kind of information sent between components of the system.

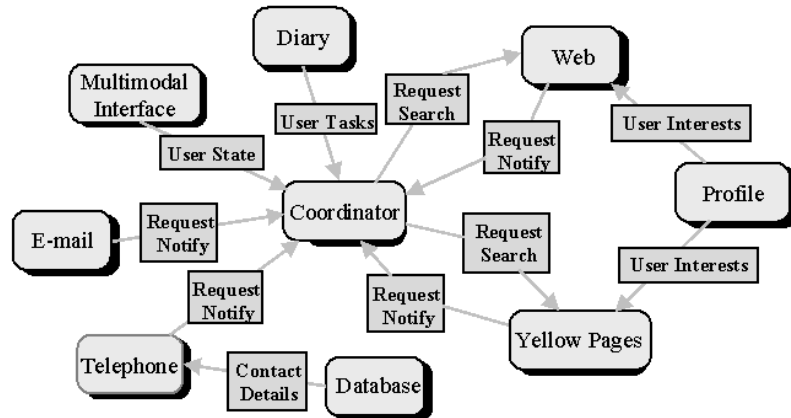


Fig. 8. Agent Interactions

The world monitor of the Coordinator is responsible for maintaining information about the user's state and future activities. One aspect of the world state is the user's willingness to accept interruptions from the system. The Coordinator includes a simple interface allowing the user to accept or refuse all interruptions, telephone calls, and e-mail notifications; the user may change state at any time (selecting to accept/refuse all interruptions entails accepting/refusing telephone calls and e-mails). To maintain consistency, the world monitor checks this parameter every second. The system also maintains an up to date model of the user's tasks based on information from the diary. Since the diary is organized into half hour chunks, information concerning the user's interruptibility based on the diary need only be updated on the half hour.

In the IA, tasks in the user's diary, when confirmed by the user, trigger the plans of the Coordinator. We describe the simple example of a lunch booking entered in the diary for 1:00 on the current day. The plan related to the lunch booking contains three actions: (i) using the YPA to find a restaurant, (ii) finding the web page of the person having lunch with the user, and (iii) reminding the user of the lunch appointment. Suppose that the user also enters a non-interruptible 1 hour meeting at 12:00 on the current day.

The whole process starts with the user's confirmation of the diary entries, when messages are sent from the Diary Assistant to the Coordinator giving the user's latest schedule. The first action of the Coordinator is to update its world model on the basis of this information. Then the IRSA Interpreter treats some

of these user tasks, in particular the lunch booking, as goals to be acted upon. For each such goal, the interpreter retrieves the appropriate plan of actions, and calls the scheduler which creates a new system task for each action in the plan parametrized according to additional information supplied by the user concerning the user task (in this case, who the lunch appointment is with). In this example, the YPA and Web searches are scheduled for the current time, while the reminder is scheduled for 11:55 (5 minutes before the user's non-interruptible meeting), which is 5 minutes before the latest time the user can be assumed to be reachable. The IRSA Execution Module then executes both the YPA and Web Search Request actions, which simply involves sending request messages to the respective assistants.

The YPA augments the query given by the Coordinator (to find a restaurant in the user's locale) with the interests of the user (obtained from the user's profile) to search for a restaurant with a particular type of cuisine. The YPA accepts a natural language query such as, in this case, 'Italian restaurant in Ipswich'. The Web Assistant searches for the lunch partner's home page. When the YPA and Web assistants have finished searching, they must notify the user that the results are ready. However, the assistants each request permission from the Coordinator to interrupt the user, again by sending appropriate messages. Should the user be interruptible, permission will be granted by reply; assume otherwise. Then the Coordinator will schedule notifications to the user (that the output is ready) for the next available time that the user is interruptible, again based on information supplied from the diary, possibly overridden by the user via the interface. This occurs through the creation of new internal goals, then a plan being retrieved by the interpreter (as discussed above), and times for the notification tasks being set by the scheduler. Finally, the Coordinator performs the notifications and reminders at the determined times.

An added complication is that the Coordinator user interface enables the user to override the system's current information concerning interruptibility (when there is no entry in the diary covering a particular time, the user is assumed to be interruptible). Supposing this has been done when a notification is to be executed, the scheduler is invoked to reschedule the task for the next time the user is interruptible (i.e. based on the then available information, which may be different from when the task was originally scheduled), provided also that the deadline for the task has not passed. It may be that because of this, some notifications never get issued; this is unavoidable. The way this is implemented at present is that the program for executing the notification first tests whether the user is interruptible, then if not, makes an explicit call to the scheduler to reschedule the task.

4 Conclusion and Further Work

Time management is an important and intricate aspect of the Intelligent Assistant, involving both the construction of user schedules in an intuitive and predictable way and the planning and execution of system tasks so as to minimize

demands on the user's attention and maximize the usage of system resources. The coordination of assistants is complicated by the fact that the assistants are all autonomous, i.e. techniques based on centralized, or even distributed, planning are not appropriate. We have used a novel agent architecture based on ideas from reactive scheduling as the basis of the Coordinator. IRSA extends the PRS architecture with temporally specific beliefs, goals and intentions; the IRSA interpreter is capable of scheduling its own future actions and then executing those actions at appropriate times. In addition to strengthening the temporal reasoning capability of the system, current work on the Coordinator concerns the investigation of both general and domain specific heuristics for rescheduling in dynamic settings, and in particular, in the domain of the Intelligent Assistant.

The Diary Assistant in the IA helps the user schedule tasks based on their stated preferences. Preferences for start times, durations and deadlines can be specified using simple natural language terms such as *morning*, *afternoon*, etc., which are interpreted using predefined fuzzy functions, enhancing usability. The system includes two scheduling algorithms: a *global* scheduler allocates timeslots to a set of tasks so as to maximize overall satisfaction of the user's preferences, and a *local* scheduler allocates a timeslot to a single task so as to minimize disruption to an existing day schedule. Thus the Diary Assistant is particularly geared towards the incremental task scheduling necessitated by repeated interactions with the user.

Three topics are the subject of current research related to the Diary Assistant. First, we intend to enable the user to specify plans of action, or more generally, constraints between tasks. Although this can be handled by augmenting the global and local scheduling algorithms described in this paper, the possibility of using constraint solving techniques is one that must also be considered. Second, we aim to investigate approximate scheduling, by which we mean scheduling at more than one level of granularity, and incrementally refining an approximate schedule as time progresses and more information becomes available. The intuition here is that sometimes the Diary Assistant is overly specific in allocating specific timeslots to tasks before the user is ready to commit to a final time. The idea is to develop an algorithm that can determine efficiently whether an approximate schedule is feasible or infeasible: by an *approximate schedule*, we mean an allocation of tasks to intervals at larger granularity than half-hour slots, such as days or weeks. Third, we intend to investigate learning user profiles, both the fuzzy functions representing the user's preferences for different types of task, and constraints on combining the preferences for different tasks based on the context provided by related entries in the diary.

Acknowledgements

We gratefully acknowledge the contribution to research and development on the Intelligent Assistant made by Ben Azvine, David Djian, Kwok Ching Tsui, Simon Case, Gilbert Owusu, Benoit Remael and Arash Sichanie.

References

1. Georgeff M. P. and Lansky A. L.: 'Reactive Reasoning and Planning', Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pp. 677-682 (1987).
2. Zadeh L. A.: 'Fuzzy Logic = Computing with Words', IEEE Transactions on Fuzzy Systems, Vol. 4, pp. 103-111 (1996).
3. Maes P.: 'Agents that Reduce Work and Information Overload', Communications of the ACM, Vol. 37, No. 7, pp. 31-40 (1994).
4. Maes P. and Kozierok R. A. E.: 'Learning Interface Agents', Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), pp. 459-465 (1993).
5. Dent L., Boticario J., McDermott J., Mitchell T. M. and Zabowski D. A.: 'A Personal Learning Apprentice', Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), pp. 96-103 (1992).
6. Prade H. M.: 'Using Fuzzy Set Theory in a Scheduling Problem: A Case Study', Fuzzy Sets and Systems, Vol. 2, pp. 153-165 (1979).
7. Dubois D., Fargier H. and Prade H. M.: 'Fuzzy Constraints in Job-Shop Scheduling', Journal of Intelligent Manufacturing, Vol. 6, pp. 215-234 (1995).
8. Dubois D. and Prade H. M.: 'Processing Fuzzy Temporal Knowledge', IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, pp. 729-744 (1989).
9. Kerr R. M. and Walker R. N.: 'A Job Shop Scheduling System Based on Fuzzy Arithmetic', Proceedings of the Third International Conference on Expert Systems and the Leading Edge in Production and Operations Management, pp. 433-450 (1989).
10. Slany W.: 'Scheduling as a Fuzzy Multiple Criteria Optimization Problem', Fuzzy Sets and Systems, Vol. 78, pp. 197-222 (1996).
11. Türksen I. B.: 'Scheduling System Design: Three Fuzzy Theory Approaches', in 'Fuzzy Information Engineering', eds, Dubois D., Prade H. M. and Yager R. R., John Wiley & Sons, New York (1997).
12. Lesaint D., Azarmi N., Laithwaite R. and Walker P.: 'Engineering Dynamic Scheduler for Work Manager', BT Technology Journal, Vol. 16, No. 3, pp. 16-29 (1998).
13. Smith S. F.: 'OPIS: A Methodology and Architecture for Reactive Scheduling', in 'Intelligent Scheduling', eds, Zweben M. and Fox M. S., Morgan Kaufmann, San Francisco, CA (1994).
14. Smith S. F.: 'Reactive Scheduling Systems', in 'Intelligent Scheduling Systems', eds, Brown D. and Scherer W., Kluwer Academic Publishers, Dordrecht (1994).
15. Baldwin J. F., Martin T. P. and Pilsworth B. W.: 'Fril – Fuzzy and Evidential Reasoning in Artificial Intelligence', Research Studies Press, Taunton (1995).
16. Bratman M. E.: 'Intention, Plans and Practical Reason', Harvard University Press, Cambridge, MA (1987).
17. Bratman M. E., Israel D. J. and Pollack M. E.: 'Plans and Resource-Bounded Practical Reasoning', Computational Intelligence, Vol. 4, pp. 349-355 (1988).
18. Smith R. G.: 'The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver', IEEE Transactions on Computers, Vol. C-29, pp. 1104-1113 (1980).
19. Jennings N. R.: 'Controlling Cooperative Problem Solving in Industrial Multi-agent Systems Using Joint Intentions', Artificial Intelligence, Vol. 75, pp. 195-240 (1995).
20. Sycara K. P. and Zeng D.: 'Coordination of Multiple Intelligent Software Agents', International Journal of Cooperative Information Systems, Vol. 5, No. 2, pp. 181-211 (1996).

21. Durfee E. H. and Lesser V. R.: 'Using Partial Global Plans to Coordinate Distributed Problem Solvers', Proceedings of the Tenth International Joint Conference on Artificial Intelligence, pp. 875-883 (1987).
22. Decker K. S. and Lesser V. R.: 'Designing a Family of Coordination Algorithms', Proceedings of the First International Conference on Multi-Agent Systems, pp. 73-80 (1995).
23. Firby R. J.: 'Adaptive Execution in Complex Dynamic Worlds', Yale University Technical Report YALEU/CSD/RR #672 (1989).
24. Ferguson I. A.: 'Toward an Architecture for Adaptive, Rational, Mobile Agents', in 'Decentralized AI 3', eds, Werner E. and Demazeau Y., Elsevier, Amsterdam (1992).
25. Gat E.: 'Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots', Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), pp. 809-815 (1992).
26. Lyons D. M. and Hendriks A. J.: 'A Practical Approach to Integrating Reaction and Deliberation', Proceedings of the First International Conference on AI Planning Systems (AIPS), pp. 153-162 (1992).
27. Müller J. P. and Pischel M.: 'An Architecture for Dynamically Interacting Agents', International Journal of Intelligent and Cooperative Information Systems, Vol. 3, No. 1, pp. 25-45 (1994).
28. Bonasso R. P., Kortenkamp D., Miller D. P. and Slack M.: 'Experiences with an Architecture for Intelligent, Reactive Agents', in 'Intelligent Agents II', eds, Wooldridge M. J., Müller J. P. and Tambe M., Springer-Verlag, Berlin, pp. 187-202 (1996).
29. Sloman A.: 'Exploring Design Space and Niche Space', in 'SCAI-95', eds, Aamodt A. and Komorowski J., IOS Press, Amsterdam (1995).
30. Rao A. S. and Georgeff M. P.: 'An Abstract Architecture for Rational Agents', Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), pp. 439-449 (1992).
31. Lee J., Huber M. J., Kenny P. G. and Durfee E. H.: 'UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications', Proceedings of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, pp. 842-849 (1994).
32. Nwana H. S., Ndumu D. T. and Lee L. C.: 'ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems', Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, pp. 377-391 (1998).