CMRadar: A Personal Assistant Agent for Calendar Management

Pragnesh Jay Modi, Manuela Veloso, Stephen F. Smith, Jean Oh

Department of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213
{pmodi,mmv,sfs,jeanoh}@cs.cmu.edu

Abstract. Personal assistant agents have long promised to automate routine everyday tasks in order to reduce the cognitive load on humans. One such routine task is the management of a user's calendar. In this paper, we describe CMRadar, a calendar management system that is a significant step towards achieving the enduring vision of assistant agents. CMRadar is an implemented system with wide-ranging capabilities for supporting email exchange, multiagent negotiations and schedule optimization based on user preferences. The motivation is to develop an end-to-end system for use by real users to obtain data to facilitate learning. Having now completed an initial prototype which we believe is the first end-to-end agent for calendar management, we present as contributions our architecture design, the communication language used to tie system components together, and initial simulation experiments that isolate negotiation cost a key factor to be logged and predicted in order to improve performance.

1 Introduction

One of the more compelling visions for agents research is the development of "personal assistant agents" that are tasked with making people and organizations more efficient by autonomously handling routine tasks on behalf of their users [6] [7] [3]. Most recently, several researchers including ourselves have embarked on a large research project, called The Radar Project [8], whose overall goal is to develop a personalized agent that is able to assist its user in a wide range of everyday tasks. Within this larger project, we are concerned with the more focused task of managing a user's calendar. While isolated aspects of calendar management have been investigated before [9] [10] [5] [4], in this paper, we present CMRadar, a *complete* agent with capabilities ranging across the full spectrum of calendar management, from natural language processing of incoming scheduling-related emails, to making autonomous scheduling decisions, to negotiating with other users, to user interfacing and visualization. Although many research issues remain, we believe CMRadar is the first end-to-end agent for automated calendar management.

A key contribution of the design of CMRadar is the specification of a basic representation, called a Template, for communicating calendar scheduling related information. The Template data structure is used as the language for the communication between the components in CMRadar and as the "glue" that binds them together. In addition, Templates are also used to normalize unformatted natural language emails into a machine readable format. We offer the Template data structure as a flexible approach to the general design of a meeting scheduling agent.

The CMRadar architecture contributes a modular design in which the core scheduling functions of the agent are separated from the multiagent aspects of calendar management. Rather than an approach that tightly couples schedule optimization and negotiation, CMRadar has a separate Manager component which handles the sending and receiving of messages from other agents and more generally, manages the negotiation with others. The Manager then communicates via Templates with a separate Scheduler component that handles the core optimization problems. We found that this modular architecture facilitates the integration of existing scheduling systems and indeed, a core component of CMRadar is the Ozone scheduler [11] originally designed for and used in several real-world logistics planning domains.

The primary underlying emphasis of the Radar project is to learn to improve performance, adapt to unexpected situations and to customize to different users. The emphasis on learning is reflected in our design of the CMRadar architecture in which all components read and write data to a central knowledge base that can be used by a separate learning process to provide feedback to the decision making components (see Figure 1). Indeed, it is the need to collect real-world data to support learning that drives our development of a complete end-to-end agent.

In anticipation of learning, this paper also presents simulation experiments in which we isolate negotiation cost as a key factor that should be logged in order to facilitate the construction of high quality schedules while avoiding high negotiation costs. We present empirical results in which an agent that remembers negotiation costs and takes them into account when deciding whether to bump a meeting outperforms more simple approaches.

2 The CMRadar Agent

CMRadar is developed as a personalized agent that interacts with other users or agents. Figure 1 shows the overall architecture of CMRadar with its functional modules. Dotted lines represent components not yet implemented. We present two main modules in the coming sections, namely the Manager and the Sched-

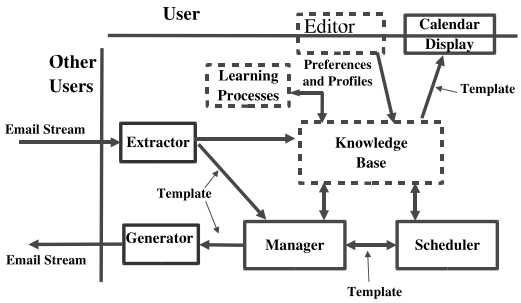

Fig. 1. Architecture of a single user's CMRadar

uler. In this section, we overview the complete architecture, briefly describing each of the modules.

Extractor: We assume that multiagent interaction in calendar meeting scheduling occurs through email message exchange. The Extractor is responsible for parsing email messages into a *template* normalized format representing the meeting request or reply to a request (see Section 4). The email messages can be sent directly by other Radar agents or by users in natural language. We have followed research on applying state-of-the-art natural language parsing techniques,[1] as well as successfully defining and applying special purpose parsing rules for language specific to meeting scheduling.[2] Figure 2 shows examples of such domain-specific language parsing rules.

duration: ["for" | "last"] <digits> ["hr(s)" | "min(s)"]
timeslots: ["at" | "before" | "after"] <time-exp>
: <time-exp> "to" <time-exp>
: <time-exp> "-" <time-exp> //*ex: "10:00 - 11:00"*
time-exp: <1-2 digits>":"<2 digits> //*ex: "10:00", "6:00"*
: <1-2 digits>":"<2 digits> <tag>
: <1-2 digits> <tag> // *ex: "10 am", "1 pm"*
tag: ["am" | "a.m." | "pm" | "p.m."]

Fig. 2. Example grammar rules for converting emails to Templates.

Manager: Calendar management is in its essence a multiagent problem as meetings involve more than one person. The Manager module in CMRadar

[1] We thank Donna Gates, Lori Levin, and Benjamin Han for their NLP work.
[2] We thank Kerry Hannan for her NLP work.

explicitly handles the multiagent aspects of calendar management, including flexible negotiation with other agents and control of email threads. Meeting scheduling is a complex process dependent on many factors, and different users schedule meetings with other users according to many different strategies. We view this variety of possible multiagent interactions similar to a *playbook* approach that we have previously developed in robot soccer [2]. The Manager can represent and reason about several different multiagent (team) strategies and learn to select the ones that are more effective when interacting with other specific agents.

Scheduler: The core task of meeting scheduling involves determining times for the meetings. The Scheduler module in CMRadar handles all the time analysis. It receives (or initiates) a specific request for a meeting and returns the user's time availability by considering the user's preferences and its calendar with different kinds of commitments. Calendar management is handled by the Scheduler under a rich set of soft and hard constraints, and agents can reason truthfully and rationally about their preferences towards optimizing the general social welfare. [3]

Calendar Data and Display: A human user is used to maintaining a calendar using the existing available COTS calendars. Our Calendar Data and Display module aims at having CMRadar use the same calendar programs. The current system is integrated with MS Outlook as shown in Figure 3. [4]

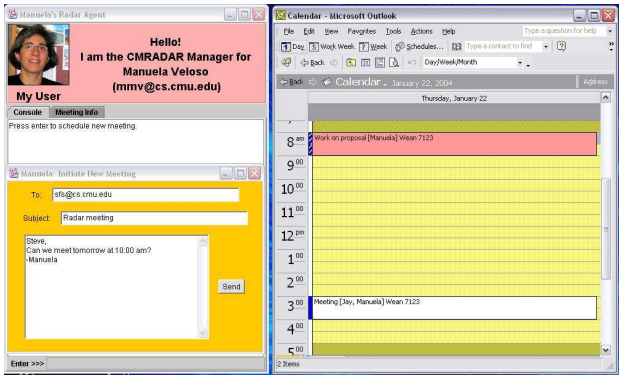

Fig. 3. A single user's CMRadar integrated with MS Outlook

[3] We thank Elisabeth Crawford for her initial work on addressing this problem under a game theoretical approach.

[4] We thank Andrew Faulring and Brad Meyers for the integration of CMRadar with MS Outlook.

Learning Although not yet part of the system, learning is necessary because obtaining ill-structured user preferences and customizing behavior to different users by hand is infeasible. The agent must acquire much of its required knowledge about its specific user over time through experience. The current CMRadar, as we present in this paper, is the base step towards a complete CMRadar agent which will be truly a learning agent.

3 Template Data Format

A Template is a special-purpose language we have constructed for all the multi-agent communication related to meeting scheduling in CMRadar. For communication with other agents or humans, templates are converted to and from natural language emails using the Extractor and Generator. We have found the Template representation to be extremely flexible for not only inter-agent communication but also for gluing components together within an agent.

As shown in Figure 4, a Template consists of a number of fields and we discuss three of the main ones: *timeslots*, *attendants* and *purposes*.

- **Timeslots:** This is a list of individual time slots, each of which contains a feasible scheduling window, denoted by earliest-start-time and latest-finish-time, and a specific desired slot denoted by start-time and finish-time. The priority field is the preference for this slot relative to other time slots in the template. Each time slot has an associated status field whose value is taken from {possible,impossible,pending,confirmed}. The "(im)possible" value is used to indicate the (un)availability of a time slot, "pending" indicates that a time slot is currently reserved by an on-going negotiation and finally, "confirmed" indicates that the meeting has been scheduled and the time slot cannot be used for another meeting without triggering a rescheduling negotiation.
- **Attendants:** This is a list of all participants of the meeting. Each attendee has an associated priority level that can be used to indicate the person's relative importance to the meeting.
- **Purposes:** This field is used to hold general text related to the meeting. Of note is the predefined-kind field which references an existing taxonomy of meetings in the knowledge base, e.g., project-meeting, faculty-meeting, advisor-meeting, etc. This field is used by the Scheduler to inform scheduling decisions, e.g., prefer to schedule faculty-meetings on Friday.

We have found in the development of CMRadar that this template-based communication is notable for its flexibility where a Template can be interpreted according to the context of the current negotiation. For example, when an agent

receives a new template containing a time slot with status “pending”, it is interpreted as a proposal to meet at that time. Conversely, when an agent receives a template that is a reply to a previous proposal containing a time slot with status “pending”, the agent interprets it as an affirmative reply to its previous request. Templates are currently used for single occurrence meetings. Representation of *repeating* meetings, i.e., “Every Monday at 2:00 pm” is not yet available, but we envision extending the template to define a set of keywords such as “every”, “every other”, “monthly”, etc, to represent this type of meeting.

```
(template
  (meeting-id MT5) (msg-id MGS1205)
  (timestamp 2003-12-17[15:04 -0500])
  (initiator sfs@cs.cmu.edu)
  (duration 3600) (location NSH1305)
  (time-slots
    (time-slot
      (earliest-start-time 2003-12-17[15:00 -0500])
      (latest-finish-time 2003-12-17[16:00 -0500])
      (start-time 2003-12-17[15:00 -0500])
      (finish-time 2003-12-17[16:00 -0500])
      (priority 1)
      (status confirmed)))
  (attendants
    (attendant (id sfs@cs.cmu.edu) (level 1.0))
    (attendant (id mmv@cs.cmu.edu) (level 1.0)))
  (purposes
    (purpose (predefined-kind project-meeting)
          (description “Radar project meeting”)
          (special-note nil))))
```

Fig. 4. Example of a CMRadar template

4 Manager

The purpose of the manager is to interact with other users and agents in service of scheduling meetings. These interactions can be very complex requiring sophisticated decision making by the agent. To illustrate the complexity of the problem, we describe a meeting scheduling episode between four users shown in Figure 5 which highlights many of the key decisions to be made by an agent.

The Y-axis in Figure 5 shows four users each with a level of priority as shown in parentheses. The priority of a user represents his or her rank relative to others in the organization. At time t1, User 3 proposes (via an email message) to User 4 a meeting M1 at 10 am. Although the calendars of each agent are

not shown, let us assume that User 4 has a meeting M2 already confirmed with Users 1 and 2 for 10 am. Because User 3 has a high priority, User 4 wishes to accommodate User 3's request and so tries to reschedule meeting M2 to another time. At t2 and t3, User 4 sends proposals to reschedule M2 for 12 pm. At t4, we see that User 1 enters into negotiations with others to attempt to accommodate User 4's request, while at t5, User 2 responds to User 4. At t6, User 4 receives a request for a new meeting. At t7, User 1 responds to User 4 by counter proposing a range of times, indicated preferences but not a hard constraint. User 4 chooses a time within the given range and re-proposes to User 2 (the other attendee of M2). At t9, User 2 responds affirmatively and at times t10,t11,t12 User 1 is able to finally confirm both meetings M1 and M2.

To successfully execute this scenario requires the following key competencies.

- **Multiple Thread Management:** A key challenge is the management of multiple inter-linked negotiation threads. Indeed, performing such "multi-linked negotiations" [12] effectively is an outstanding research issue in multiagent systems. As illustrated by the interactions between meeting M1 and M2 in our scenario, the decision to accept a meeting proposal for a given time may depend on the result of other negotiations. Furthermore, others may not respond immediately to requests or may never respond at all. The agent must be able to keep track of the status of different negotiation threads and their interactions and make timely decisions.
- **Context-Dependent Negotiation Strategies:** The agent must be flexible and able to adapt its decision making depending on the context of the current meeting negotiation and its participants. For example, we saw that User 4 chose to accommodate User 3's request by bumping meeting M2 which resulted in a renegotiation with others. In other scenarios, it may be better to refuse User 3's proposal. Indeed, this type of decision cannot be a static one but must take into account the current context. The appropriate strategy to use in a given situation depends on very rich context information including local user preferences, the other participants of the meeting, the history of the negotiations with those people, and the history of the current negotiation itself.
- **Explanation:** The agent must be able to keep the user informed of the status of on-going negotiations and explain scheduling decisions when asked. For example in our scenario, if User 4 queries her agent as to why meeting M2 has been rescheduled, the agent should be able to respond that it was due to request for a meeting M1 from User 3. This competency is crucial for usability and trust by humans.

In our current system, the Manager responds to requests for meetings on its user's behalf by querying the Scheduler (described in the next section) for free time slots. The Manager can display and update the status of meetings via the MS Outlook Calendar interface. The Manager currently uses a simple fixed negotiation strategy in which an initiator always proposes a single time slot and a receiver either accepts it or rejects it in which case the initiator re-proposes. Limited forms of rescheduling decisions are made where lower priority meetings are bumped in favor of higher priority ones. In next steps, we will expand these capabilities with more sophisticated negotiation strategies and learn to adaptively choose the appropriate negotiation strategy using a playbook strategy [2].

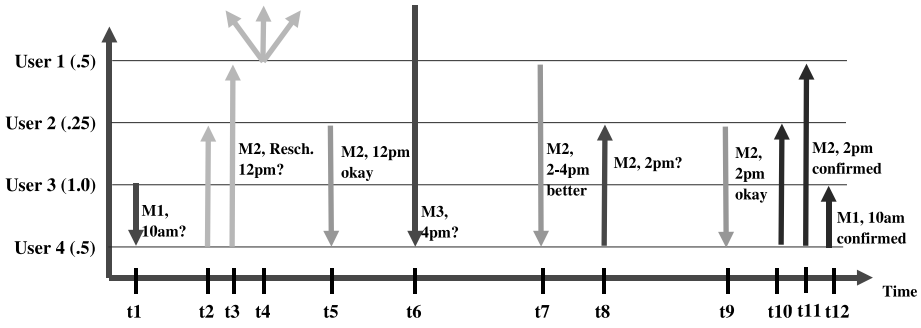

Fig. 5. Example of a complex negotiation

5 Scheduler

The Scheduler is the component within CMRadar responsible for representing and managing the user's calendar. Whereas the Manager handles the negotiation with other calendar agents, the scheduler reasons about the user's constraints and preferences to determine the best options in a given meeting context. The CMRadar Scheduler has been built using the Ozone [11], an incremental, constraint-based scheduling framework previously used to develop a number of complex logistics planning applications [1]. Ozone is designed specifically from a continuous scheduling mindset, where schedules evolve incrementally over time as new requirements are received, priorities change and unexpected conflicts arise. This orientation makes it ideally suited for the problem of calendar management.

Within CMRadar, the Scheduler provides basic support for both (1) responding to meeting requests and (2) assisting in initiation of meeting requests. In both cases, the action of the Scheduler is to generate options that are consistent with the constraints specified in the triggering message and maximize satisfaction of known user preferences. In more detail, the Scheduler's response to a meeting request (originating either from another agent or the user herself) proceeds in three steps (see Figure 6):

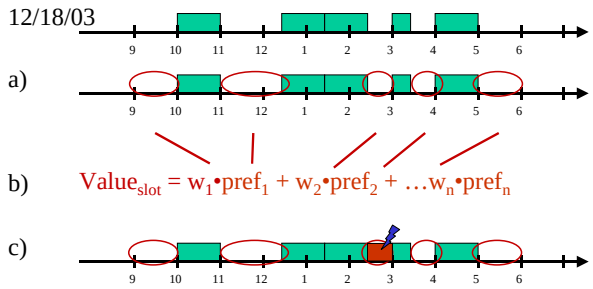

Fig. 6. Responding to a Request

- **Generate feasible meeting options:** The input meeting request template specifies a set of acceptable time periods (in the simplest case, an earliest start time est_m and a latest finish time lft_m) together with any constraints on meeting duration. By default, a feasible option is any time slot that satisfies these constraints and is currently not booked for another purpose. If these constraints yield no options, the Scheduler will relax the constraint that existing meetings must be respected, and consider pre-emption of lower priority meetings (currently a function of attendees and meeting type). Other more complex resolution strategies (e.g., shrinking the durations of one or more existing meetings) are also possible.
- **Collect and evaluate preferences:** The scheduler maintains representations of various meeting attendants (including the user) , meeting groups, meeting types, etc., which provides a backbone for organizing known meeting preferences. Using the parameters of the template as indicies into this representation, the set of preferences relevant to the request are collected. Each option is then evaluated and assigned a rating (discussed in more detail below), indicative of how well each satisfies this set of preferences.
- **Select option(s):** According to the current response strategy in force, the highest valued option or the n highest valued options are returned.

As suggested above, a preference assigns a *utility* to a given meeting option, reflective of how well it is satisfied. Utilities are defined to span the range $[-1, 1]$, with 1 implying that the preference is completely satisfied, 0 that it is neutral, and -1 indicating that the option is intolerable. A preference also has an intrinsic *importance* in relation to other preferences, a number between 0 and 1. In a given option evaluation context, the importance values of all relevant preferences are normalized to produce the set of weights for computing the overall rating of a given option (see Figure 6). One aspect of this normalization involves striking a balance between those preferences held by the user and those held by other meeting participants.

The set of preferences that must be accounted for in meeting scheduling are quite diverse, and we have devised representations that allow specification of several broad classes. In addition to simple interval preferences (e.g., I prefer to meet from 2:00PM to 3:00PM, piece-wise linear curves can be use to specify more complex time-of-day preferences (e.g., afternoon is best, late morning is acceptable but never before 9:00 AM). More interesting are so-called *dynamic* preferences, which depend on the current state of the user's calendar and change as the calendar schedule evolves.

Figure 7 gives an example of a dynamic scheduling preference - a preference for scheduling meetings back to back. On the left, the utility curve of the preference is shown, indicating the level of satisfaction of the preference to be a function of the percentage of total meetings over a given horizon that fall adjacent to the option (i.e., time slot) under consideration. (In this case, adjacency is defined to tolerate some, presumably small, time gaps.) On the right of Figure 7, a three meeting schedule is displayed to show the preference's utility for different options. The preference value for the time slot between Meeting 1 and Meeting 2 is highest because it will make all four meetings back to back.

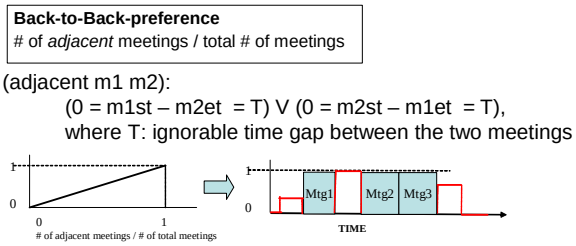

Fig. 7. Back-to-Back Preference

6 Empirical Results in Simulation

We investigate the effect of different scheduling strategies on system performance in a simulation setting. The purpose of these experiments is to provide guidance in further developing the CMRadar system by answering the question: What data should be logged by a deployed CMRadar agent in order to help it learn? Although a working prototype of CMRadar is completed as described in this paper, we use simulations for our investigation because it allows large numbers of experiments to be done efficiently. Furthermore, CMRadar has not yet been deployed to users so that in-situ studies may be done.

In the next section, we describe the experimental setup and the results obtained. The results show that a scheduling strategy that explicitly takes into account negotiation costs outperforms other simpler strategies assuming that these costs can be perfectly known. Of course, predicting negotiation costs can be a difficult challenge in the real world and is one of the key issues that we will investigate next. However, the results presented in this section are significant because they show that recording past negotiation costs and learning to predict them would be valuable since they can be used to improve system performance.

6.1 Experimental Setup

While CMRadar is responsible for making many different calendar management decisions, this section focuses on one key decision related to meeting scheduling in which the agent must decide in which time slot to put a particular meeting. This decision is complicated by the fact that multiple meetings may compete for the same time slot. For example as we saw in Figure 5, M2 occupied a time slot (10 am) that was desired by a new higher priority meeting M1. The agent must decide whether to assign the time slot to M1 and "bump" M2 and reschedule it, or keep M2 where it is and put M1 somewhere else. The decision to bump M2 incurs cost because it requires sending messages to the other attendees with a request to reschedule and waiting for responses. These responses may not come immediately because the other attendees may in turn need to bump other meetings. If these costs are large, it may be better for the agent to not bump M2 and instead find an alternative time slot for M1. Note that it is difficult to determine in advance which is the better decision because the costs incurred for each decision are not known and because other people's schedules and preferences are not directly observable.

To investigate this issue, we use the following model. Let $calendar = \{slot_1, slot_2, ..., slot_n\}$ be the set of time slots in a user calendar and $M = \{M_1, M_2, ..., M_m\}$ be a set of meetings. The task of the agent is to determine a schedule in which no more than one meeting is in a given time slot. Let the

function $sched : calendar \rightarrow M$ be a schedule that maps time slots to meetings. For each meeting, the user has a preference ordering over all time slots. Let $V_p^i(M_j)$ denote the user preference for putting meeting M_j in $slot_i$. For a given schedule $sched$, let $V_p(sched) = \sum_{slot_i \in calendar} V_p^i(sched(slot_i) = M_j)$ denote the total quality of the schedule. Finally, the assignment of a meeting to a time slot incurs some negotiation cost which can be a complex function of the other attendees of the meeting and the time slot. Let $C_n^i(M_j)$ be the cost for putting meeting M_j in $slot_i$. We assume that each meeting has a unique desired slot where there is zero cost if it is scheduled in that slot. The "desired slot" models a time that is proposed by other attendees. That is, when an agent receives a request "Can you meet at 10 am?", there is no negotiation cost for scheduling at 10 am because presumably the requester is free at that time. So we assume $C_n^i(M_j) = 0$ if $slot_i$ is the desired slot for M_j. Finally, the total cost of a schedule is given by $C_n(sched) = \sum_{slot_i \in calendar} C_n^i(sched(slot_i) = M_j)$. While this model is clearly limited in many respects, it provides a simple yet effective approach for investigating alternative decision-making strategies.

6.2 Evaluating Strategies

Using the above model, we now present empirical results for three different decision-making strategies: *Greedy*, *Bumping* and *Ncost*. In the simplest Greedy strategy shown in Figure 8, an agent only inserts meetings into free slots and never backtracks on these decisions. If the desired slot for a particular meeting is already taken, the meeting is inserted into an alternative time slot that is both free and highest-ranked according to local preferences. For not putting the meeting in the desired slot, the agent incurs a negotiation cost as shown in line 6 of Figure 8.

```
procedure GreedyStrategy(M_i, desiredSlot)
(1)   if desiredSlot is not null and is free:
(2)     put M_i into desiredSlot
(3)   else:
(4)     timeslot ← best free slot for M_i
          according to preferences
(5)     put M_i into timeslot
(6)     c_i ← actual cost of negotiating with
          other attendees for M_i
(7)     C_n = C_n + c_i
```

Fig. 8. Greedy Meeting Scheduling

The Bumping strategy shown in Figure 9 is more complex because it bumps meetings out of their desired slots if a new meeting is more preferred. However, the decision to bump or not is made exclusively using local preferences and does not take into account negotiation costs that may be incurred.

```
procedure BumpingStrategy(M_i, desiredSlot)
(1)   if desiredSlot is free:
(2)     put M_i into desiredSlot
(3)   else:
(4)     M_j ← current meeting in desiredSlot
(5)     v_j ← preference for M_j in desiredSlot
(6)     v_i ← preference for M_i in desiredSlot
(7)     if v_i > v_j: // bump M_j
(8)       put M_i into desiredSlot
(9)       GreedyStrategy(M_j, null)
(10)    else:
(11)      GreedyStrategy(M_i, null)
```

Fig. 9. Scheduling with Bumping

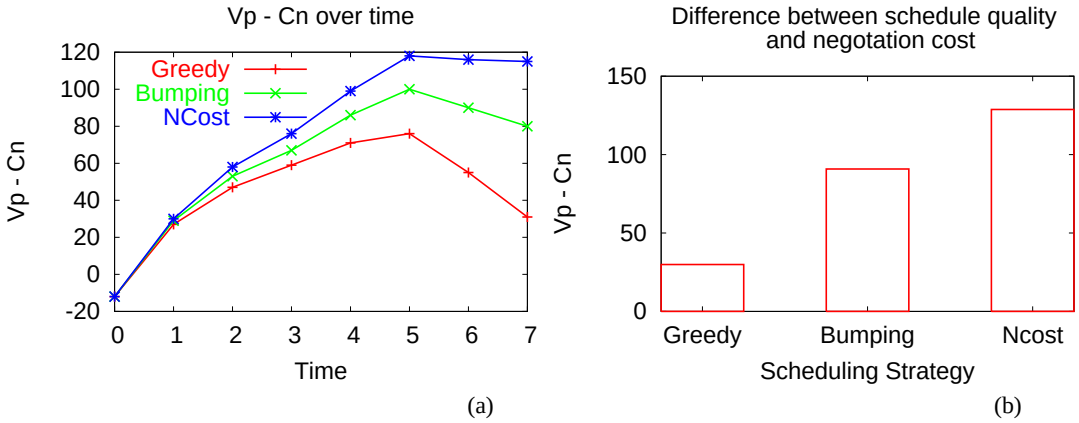

Fig. 10. The NCostStrategy, which trades off local preference (V_p) with negotiation cost (C_n), outperforms a strategy that schedules only in free slots (Greedy) and one that only uses preferences (Bumping). (b) shows the final V_p - C_n for the time series data shown on (a).

Finally, the NCost strategy shown in Figure 11 takes into account both local preferences and predicted negotiation costs. As shown in line 9, an existing meeting M_j is bumped in favor of meeting M_i only if the preference for M_i

minus the cost for renegotiating for M_j is greater than preference for M_j minus the cost for renegotiating for M_i.

```
procedure NCostStrategy(M_i, desiredSlot)
(1)   if desiredSlot is free:
(2)     put M_i into M_i
(3)   else:
(4)     M_j ← current meeting in desiredSlot
(5)     v_j ← preference for M_j in desiredSlot
(6)     v_i ← preference for M_i in desiredSlot
(7)     c_i ← predicted cost of negotiating with
          other attendees for M_i
(8)     c_j ← predicted cost of negotiating with
          other attendees for M_j
(9)     if v_i − c_j > v_j − c_i: // bump M_j
(10)      put M_i into desiredSlot
(11)      GreedyStrategy(M_j, null)
(12)    else:
(13)      GreedyStrategy(M_i, null)
```

Fig. 11. Scheduling with consideration of negotiation costs

Figure 10 shows the empirical results averaged over 100 runs. In each run, we use a calendar consisting of 9 one hour time slots (from 8 am to 5 pm) initialized with 25% of the slots full. The agent is tasked with iteratively scheduling meetings as they arise over time. The negotiation cost for a particular meeting is modeled as a random number taken from [0,100]. The user preference $V_p^i(M_j)$ is calculated as the priority of M_j divided by the distance of $slot_i$ from $slot_{pref}$, where priority is a random number taken from [0,100] and $slot_{pref}$ is a random slot in the calendar. Thus V_p and C_n range from zero to 100 times the number of meetings in the calendar. Figure 10 (a) shows how performance varies as 7 meetings are scheduled in sequence, while Figure 10 (b) shows that the final schedule obtained using NCost strategy is superior in performance to the others as measured by the quality of the resulting schedule (V_p) minus the negotiation costs for obtaining that schedule (C_n). These results indicate that logging past negotation costs and using them to predict future costs can be used to improve performance.

7 Conclusion

We presented CMRadar, an implemented system for calendar management. CMRadar is a complete agent in the sense that it is able to facilitate meeting scheduling across the entire spectrum from initiation to confirmation to rescheduling.

While we have thus far concentrated on breadth of functionality, in future work we will develop more sophisticated reasoning within each component along the dimensions of the key challenges discussed. CMRadar indeed presents a challenging road map for future research in effective learning personal assistant agents.

Acknowledgments

This work is supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHC030029. We thank the following people for their work on CMRadar: Andrew Faulring, Brad Meyers, Kerry Hannan, Lawrence Lee, Akiva Leffert, Elisabeth Crawford, Donna Gates, Benjamin Han, and Lori Levin.

References

1. M.A Becker and S.F Smith. Mixed-initiative resource management: The amc barrel allocator. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, pages 32–41, Breckenridge CO, April 2000. The AAAI Press.
2. Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004.
3. H. Chalupsky, Y. Gil, C.A. Knoblock, K. Lerman, J. Oh, D.V. Pynadath, T.A. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. In *Proceedings of Innovative Applications of Artificial Intelligence Conference*, 2001.
4. Leonardo Garrido and Katia Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*.
5. N. R. Jennings and A. J. Jackson. Agent based meeting scheduling: A design and implementation. *IEE Electronics Letters*, 31(5):350–352, 1995.
6. Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7), 1994.
7. Tom M. Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, 1994.
8. The Radar Project. "www.radar.cs.cmu.edu", 2004.
9. Sandip Sen and Edmund H. Durfee. On the design of an adaptive meeting scheduler. In *Proc. The Tenth IEEE Conference on Artificial Intelligence for Applications*, pages 40–46, 1994.
10. Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling. In *Group Decision and Negotiation*, volume 7, pages 265–289, 1998.
11. S.F. Smith, O. Lassila, and M.A. Becker. Configurable, mixed-initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, 1996.
12. Xiaoqin Zhang and Victor Lesser. Multi-linked negotiation in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, 2002.