An automated distributed meeting scheduler

Sandip Sen
Department of Mathematical & Computer Sciences
University of Tulsa,
600 South College Avenue,
Tulsa, OK 74133
e-mail: sandip@kolkata.mcs.utulsa.edu

Introduction

A central thesis in our research has been the claim that a number of routine information processing needs in organizations can be efficiently automated. As such, we are interested in designing and implementing software systems that automate and share information processing tasks of associated human users. The benefit of such software is two-fold: they allow users to concentrate on more productive tasks, and they improve the quality of information processing by preventing errors that might be introduced by human users due to the routine and tedious nature of the job in question.

In particular, we have studied the problem of efficiently automating the process of scheduling meetings between employees in an organization. Our approach to meeting scheduling, in contrast to most of the currently available software for centralized calendar management and meeting scheduling, is a distributed one, where each employee in the organization is provided with an automated (computational) meeting scheduling agent. When a user wants to schedule a meeting with other users, he/she inputs a meeting request to the associated meeting scheduling agent. This agent negotiates with the agents corresponding to the other users to schedule the meeting. Since all meeting requests and calendar accesses are routed through the meeting scheduling agent, it can protect the privacy of its associated user while following other preferences of this individual. The meeting scheduling agent uses the calendar manager software to manipulate the user's calendar, and uses the e-mail system to communicate messages with other meeting scheduling agents (see Figure 1).

The central question in the design of a successful meeting scheduler is the following: how can a meeting scheduling agent efficiently negotiate with other meeting scheduling agents without compromising any of the constraints specified by its associated user. In our previous work [1, 2, 3] we have studied the usefulness of different heuristic negotiation strategies to solve a precisely defined distributed meeting scheduling (DMS) problem. We developed analytical models of expected performance of heuristic strategy combinations, and verified these expectations with experiments on simulated systems.

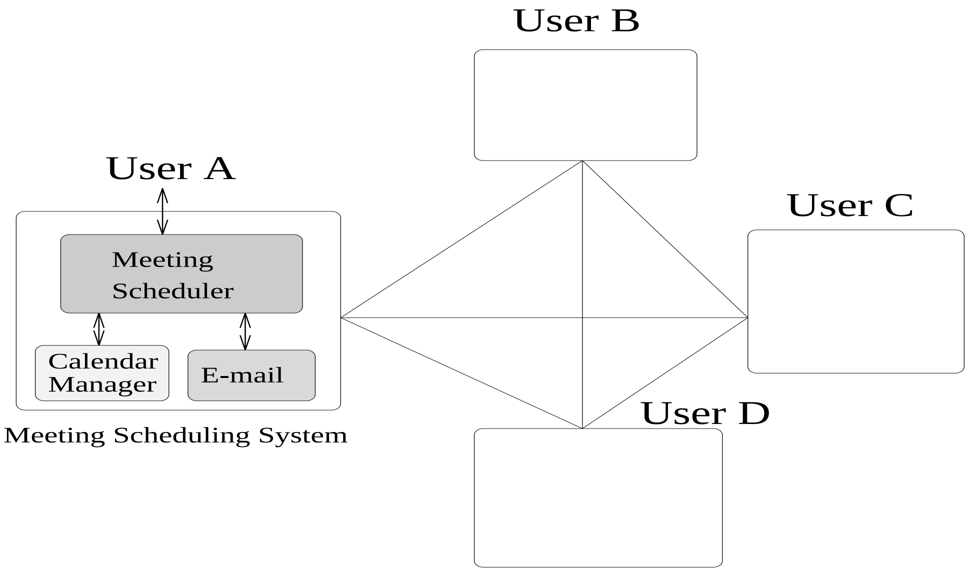


Figure 1: Distributed meeting scheduling over a network of computers.

Intelligent information agents in organizations

Computer networks that support human organizations provide an infrastructure for improving group performance through an array of collaboration tools, such as electronic mail systems and shared file systems. While such tools help people share, access, and manipulate more information, they can also impair human performance through overuse or abuse from the propagation of unnecessary information. Techniques from artificial intelligence can introduce "intelligent agents" into organizational computing systems, where these agents use knowledge about the interests and priorities of people to perform routine organizational tasks such as automatically screening, directing, and even responding to information [4]. The usefulness of intelligent agents is further appreciated with the advent of the information superhighway [5].

Some of the people in a typical organization spend a large percentage of their working time in scheduling and attending meetings [6, 7]. The efficiency of the scheduling process used and the quality of the schedules generated, then, affects the working of an organization to a large extent. Even when everyone involved in a meeting has available times to meet, the process of searching for a commonly available time in the presence of communication delays (either through electronic mail or in contacting by phone), and in the presence of other meetings being scheduled concurrently, can be frustrating and lead to less than satisfactory solutions. Automating meeting scheduling is important, therefore, not only because it can save time and effort on the part of humans, but also because this may lead to more efficient schedules and to changes in how information is exchanged within organizations. Past efforts [8, 4] in developing automated meeting schedulers have met with limited success, although they are available in a number of office software systems [9].

Humans exhibit wide variations in how they manage their calendars. The variations range over the length of the calendars used by individuals, the number of calendars used by each person, level of privacy desired, portability and accessibility requirements, percentage of scheduled meetings that are canceled and rescheduled, as well as the nature of archiving, querying, and inserting related information into the calendar. In order for an automated calendar management system to be accepted, it has to be flexible to the varying needs of the user. Such a requirement bodes well for our distributed formulation of the problem, as each user can modify his/her associated agent according to his/her own liking.

Most of the commercially available software for scheduling over computer networks have been personal computer based systems. A good survey of the most promising of these products can be found in [10]. This survey evaluates and compares the features offered by the four most powerful schedule sharing systems in practice today. These four products are: ON Technology's Meeting Maker 1.5, Microsoft's Schedule+ 1.0, Now Software's Now Up-to-Date 2.0, and WordPerfect Office 3.0's Calendar module. The most important thing to clarify here from our point of view is that none of these systems are "automatic schedulers". These products provide the users with only a nice interface to view their own calendars and that of other users, and in some cases find time intervals to propose by searching these calendars. When using these systems, users have to allow complete access to their calendars by all other users. The only restriction is that the user (or a proxy) has the sole authority to modify his/her associated calendar. As the title of the paper itself suggests, these products are designed to "share" schedules. Similarly, in the AI community, there has been a number of recent projects [11, 12] which assist a human user in scheduling meetings. None of these, however, completely automate the meeting scheduling process.

We, however, propose to provide autonomous scheduling capabilities through restricted information exchange between intelligent meeting scheduling agents. Our approach is considerably more difficult because of the significantly limited information available to scheduling agents about the state of the calendar of other users. As such, multiple rounds of information exchange between scheduling agents are often required to locate a time interval in which a requested meeting can be scheduled. The latter mode of meeting scheduling, however, does not require the intervention of the human user for each round of proposal exchange, and is likely to be significantly quicker in scheduling meetings. In addition, our proposal for autonomous scheduling can approximate the privacy and security concerns of users as specified by user-modifiable constraints (these concerns are severely compromised in the above-mentioned systems as others can freely access the user's calendar state). The distributed nature of our system allows for better throughput through concurrent negotiation on multiple meetings, and better fault tolerance.

Distributed Meeting Scheduling

When a user requests a meeting to be scheduled with other users, the associated meeting scheduling agent is designated as the *host* agent for that meeting; the agents corresponding to the other users attending the meeting are called *invitee* agents. A meeting is specified by a number of parameters:

- set of attendees,
- proposed length of the meeting,
- priority assigned to the meeting,

- a set of possible starting times on the calendar for the meeting (e.g., sometime next week, Friday afternoon, etc.),
- a scheduling deadline,
- any additional constraints.

Space does not permit an enumeration of the constraints our system can handle. Some of the typical constraints that can be handled include scheduling a meeting before/after another meeting, scheduling a meeting only if another meeting is scheduled, scheduling with a subset of attendees or for a shorter length if a meeting time satisfying all the given parameters cannot be found, scheduling repeating events, etc.

Our scheduling agents use contract-based negotiation [13] to find mutually acceptable time slots for meetings. While the contracting framework does not capture some of the sophistication that people might employ in exceptional circumstances, it balances the need to have a flexible routine to handle a range of situations, while still keeping the routine well-defined and understandable enough to be embraced by a user.

We will use an example to illustrate the workings of our proposed scheduler. Let us consider a simple meeting scheduling scenario in which user A wants to meet with users B and C for an hour sometime in the next week. When user A instructs its meeting scheduling agent (in the following we name the agents after their user) to schedule this meeting, agent A will search the local calendar to find some hour-long free slots next week that does not conflict with user A's preferences (for example, no meeting before 9 or just after lunch hours). Some of these free slots are then proposed via electronic mail to agents corresponding to users B and C for a meeting (for example, agent A may propose to meet either at 10am on Monday or at 2pm on Wednesday). If agents B and C find some of these proposals agreeable (corresponding slots are free in the calendar of their respective users, and does not violate any constraints/preferences), they respond positively by electronic mail. Alternatively, they may propose some other hours as dictated by local calendars and user preferences. These negotiation cycle is repeated until a slot is found which is agreeable to all three agents, and the meeting is scheduled for that time slot.

A simplified version of the scheduling protocol is given below:

1. When a meeting needs to be scheduled, the host tries to find time intervals in its schedule that suit the constraints of date and time. If it cannot find any interval, it fails and the meeting is abandoned. Otherwise, if it is the only participant in the meeting, it schedules the meeting for the best (earliest) interval. If there are other participants, the host announces a contract for the meeting to the invitees by proposing one or more of the best intervals found.

2. Each invitee receives the contract proposal(s), and tries to find local solutions to satisfy those contracts and send them back as bids to the host. Bids consist of time interval(s) for which the bidder (the invitee) can schedule the announced meeting. The time interval(s) sent as bids can simply be the subset of those announced by the host that the invitee has free on its calendar, or they can be counter-proposals for when to meet.

3. The host collects and evaluates these bids. If the bids suggest a common time interval which is free for the host as well, the meeting can be scheduled and the host sends awards to the bidders. If the meeting cannot be mutually scheduled yet, the host generates new proposals depending on the bids received and its own calendar and sends these off to the bidders. It also sends rejections for bids received.

4. When the bidders receive new proposals, they reply as above. On receiving an award, they check to see if those time intervals are still free. If so, they mark their calendar, recording the scheduling of the meeting. Otherwise they send back rejections.

The above algorithmic steps are repeated until a satisfactory schedule is arrived at or it is recognized that the meeting cannot be scheduled (due to an over-constrained schedule or due to the fact that the meeting could not be scheduled before its deadline).

Heuristic negotiation strategies

Though the contract net metaphor delineates the respective roles of the agents involved in a coordination process, it does not specify the local control necessary to focus the negotiation process. The most important questions that need to be answered before the contract-net protocol can be applied in the DMS domain are the following:

- How do hosts build contracts to propose?
- How do invitees bid in response to contracts?
- How do agents maintain constraints between concurrently negotiated meetings?
- How do agents handle the change in problem constraints as time passes during negotiation?

The meeting scheduling problem has been proved to be computationally intractable, and hence any realistic attempt to solve the problem has to use appropriate heuristics to find satisfactory solutions. We view the DMS problem as a distributed search problem, where each of the agents involved in a meeting searches its calendar to find a time interval for the meeting, and it is required that every attendee chooses the same time interval to schedule the meeting. To find a common free time interval of the desired length quickly, the agents must use efficient local *search biases* to guide the search process in finding suitable time intervals on the calendar to use for the meeting; decide on how much information to exchange without losing flexibility to schedule other meetings concurrently; find responses to proposals that move the search towards a solution without violating local constraints; choose whether or not to view tentative proposals as commitments; and locate already scheduled meetings to reschedule when faced with new contingencies. In the following, we briefly narrate our findings on these heuristics:

Search bias: Heuristics for *search biases* allow agents to focus search in their calendar to find preferred free intervals to schedule a meeting. A search bias provides a partial order on a given a set of alternative solutions. In the DMS domain, we have experimented with three different search biases: linear early, linear least dense, and hierarchical. Linear search biases (search biases that operate on a linear ordering of the search space) are more effective for a small number of agents working on short calendars; a hierarchical search bias using temporal abstractions of the calendar (involving months, weeks, days, hours) is more useful for larger organizations. When using linear early search bias, agents propose the earliest available time slot for a requested meeting. When using linear least dense bias, agents search the entire calendar and then propose a time interval with maximum free time around it. When using hierarchical search bias, agents scheduling a meeting first identify which is the most desirable week to schedule the meeting, followed by the most desirable day within the chosen week, and finally the particular time interval within that day. Whereas the linear early search bias produces front-loaded calendar leaving room for long meetings later on, it cannot accommodate the scheduling of meetings at short notice (and hence triggers meeting cancellations). Linear least dense and hierarchical search biases produce evenly loaded calendars that can more readily accommodate schedule disruptions, but can fail to schedule long meetings when the calendar is dense. These heuristics help agents in deciding what to propose to other agents and also to adjust their proposals as calendar states change (causing changes in problem constraints) over time.

Information exchanged: As in most problems involving distributed search, there is a tradeoff between the amount of information exchanged per iteration among the meeting schedulers and the number of iterations of information exchange needed to schedule a meeting. If everybody sent all information to one agent, then one iteration is sufficient, and we have a centralized scheduling scenario. To take full advantage of this situation, however, it is necessary to sequentialize the scheduling of all meetings, which severely affects system throughput. Limited, focused exchange of information to schedule meetings will in general require more iterations, but will allow the agents to schedule multiple meetings concurrently, thus increasing the throughput of the system. The latter form of communication will in general require less information exchange, as all the information sent by the agents in the first case may not be necessary to process a task. Furthermore, in the DMS domain, schedulers have to balance the need for efficiency (implies sending more meeting proposals per iteration) with the need for maintaining privacy (propose as few intervals as possible to schedule a meeting). Our probabilistic analysis shows that, although agents can schedule meetings quickly by sending more proposals per iteration of negotiation, the law of diminishing returns holds back the savings obtained by exchanging more information [2]. These heuristics help agents decide how much to share with other agents.

Response mechanisms: In scheduling a meeting, the invitee agent can be passive, requiring the host to do most of the work in finding a suitable contract, or be more active in guiding the negotiation through informative counter-proposals. For example, the invitee can respond by accepting or rejecting the proposal, or can counter-propose another time interval for the meeting. Cooperative revelation of local information can greatly accelerate coordination if coupled with proper local search control. We have developed analytical expectations of the savings obtained in information exchange iterations when invitee agents respond with counterproposals instead of just accepting/rejecting proposals from host agents. These heuristics address the question of how to construct bids in response to proposals.

Use of commitment: From the time an agent proposes a time interval for a meeting to the time it receives a response from the other side, other meeting requests may arrive that can use this interval. The agent has to decide whether or not to use the same (or overlapping) time intervals for simultaneous negotiations on different meetings. We have investigated how agents can reason about system load, expected coordination effort, importance of meetings being processed, etc., to make a knowledgeable decision on committing to tentative proposals [3]. These heuristics enable agents to avoid harmful interactions between concurrently negotiated meetings.

Cancellation mechanisms: When faced with changing demands on its resources, an agent may need to cancel previously scheduled meetings to accommodate new ones. A structured framework for renegging past agreements has been developed that optimizes the utility of canceling previously scheduled meetings to accommodate new ones [1]. These heuristics enable agents to rework the meeting schedules as calendar state changes with time.

Adaptive scheduling: Intelligent, autonomous agents should be able to reason and adapt to changing environmental conditions. We have developed the design of an adaptive scheduling agent that will be able to choose the most efficient local problem-solving behavior, given the current environmental parameters and the local problem-solving states of the agents involved in the negotiation process [2]. Local problem-solving behavior of the meeting scheduling agents is determined by a combination of heuristic strategy options, each option chosen from a set of alternatives available for the associated strategy dimension. Our goal has been to design the adaptive agent such that its choices are based on precise analytical expectations of the performance, as measured by some given performance metrics, of local problem-solving behavior. Agents that monitor their environment continually and adjust their behavior to suit changing demands imposed on the system can provide robust performance across a wide range of situations.

Implementation

We have implemented a distributed meeting scheduling system on a workstation based computing environment, that utilizes the above-mentioned findings. Though the current implementation runs on a local network of workstations, the negotiation mechanisms and encoding language developed can be effectively used in other types of computing environments. For example, the same techniques can be used to implement a distributed meeting scheduling system that runs on personal computers. The actual implementation will definitely be different depending on the communication and programming environment involved. As far as either the problem-solving behavior of the system or the manner in which the user interacts with the system is concerned, however, no appreciable modifications to the proposed design is required. Additionally, such a meeting scheduling system can be configured for implementation in a server-based computing environment as well (this type of environment is often used in the shared schedule implementations mentioned above). The programming language used for the current implementation is C++.

The architecture of our meeting scheduling agents is as described in Figure 2. We first briefly outline the functionalities of each of the components of the meeting scheduler, and then describe their implementations in more detail. The user interacts with the meeting scheduling system through the **user interface**. The interface allows users to input meeting requests and schedule preferences, to check the scheduling of a meeting as negotiated by

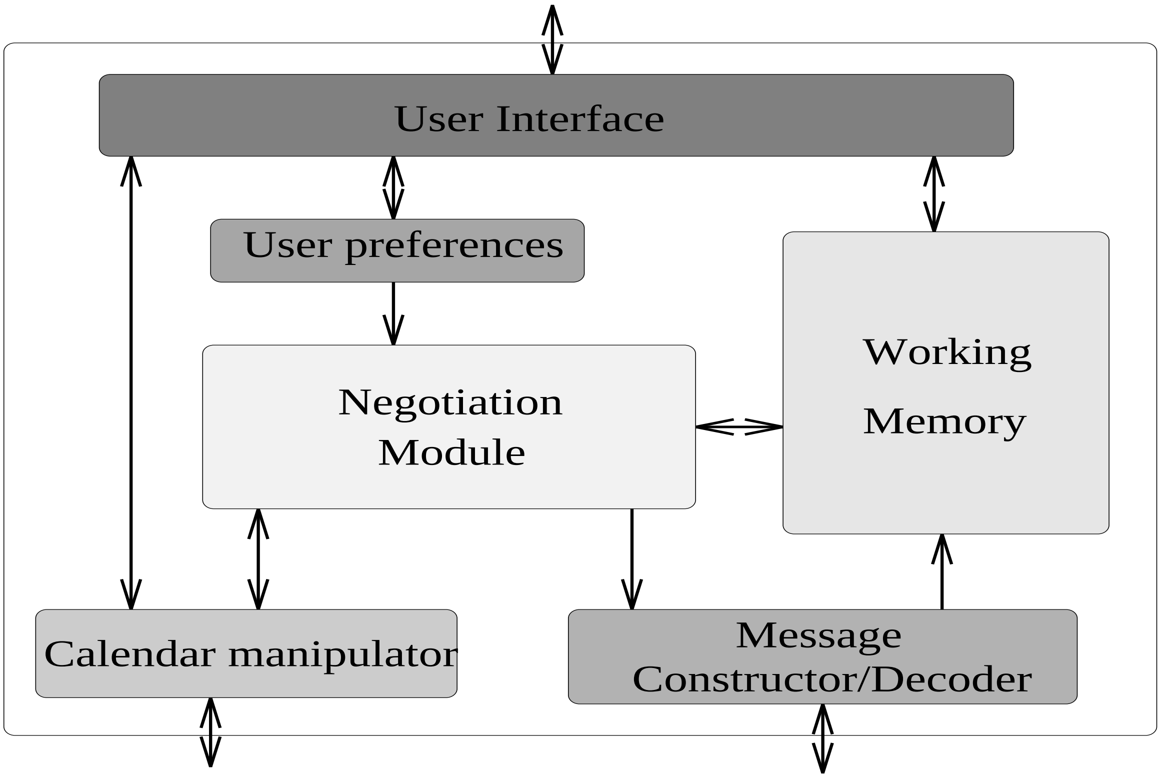


Figure 2: The architecture of the automated meeting scheduler.

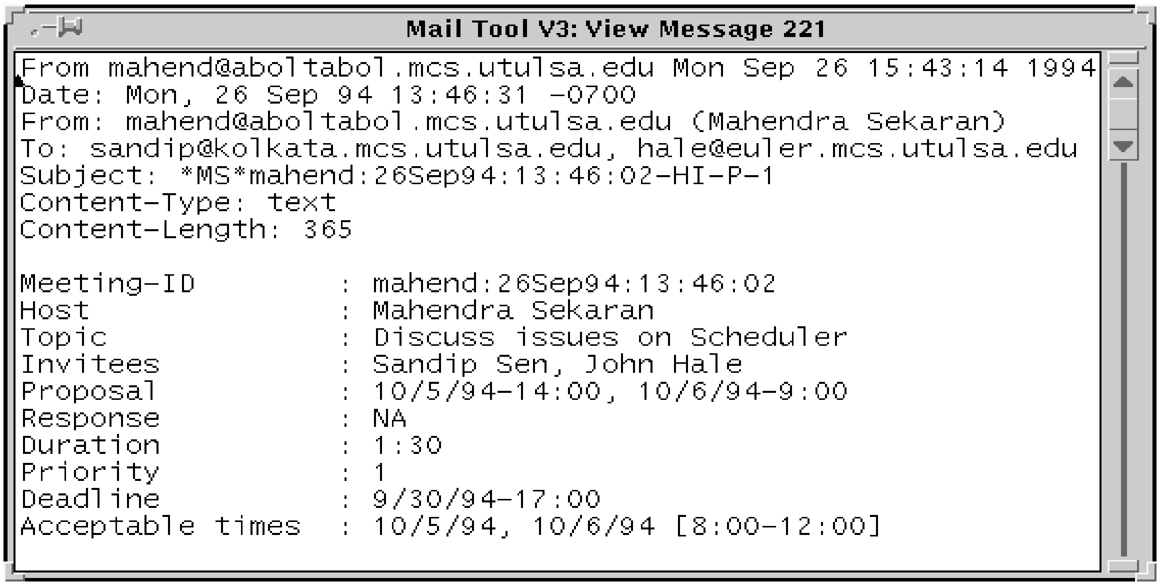
the agent, and to monitor its calendar and current negotiations. The **user preferences** component stores the preferences of the user for the nature of schedules, priorities for different types of meetings, preferences for attending meetings with other users, etc. The **working memory** contains the data structures and memory traces of negotiations on meetings that are being currently scheduled. The **negotiation module** is the "brain" of the meeting schedule system, and uses the user preferences while exchanging proposals with meeting scheduling agents representing other users to schedule meetings in the working memory. The **calendar manipulator** component allows the user interface and the negotiation module to access and modify the status of the user's schedule through the calendar management program. The **message constructor/decoder** component serves as the interface with the e-mail system through which messages are communicated with other scheduling agents. When a proposal is to be sent, this module constructs an appropriate mail message in the designated format and invokes the mailer. Similarly, when a proposal or a bid arrives from another agent, this module decodes the message and posts the proposal or bid into the working memory.

To develop a system that is portable across a number of platforms, we are using the industry-standard X Window system to develop a Graphical User Interface (GUI) through which users would interact with their meeting scheduling agent (we are using the X Toolkit consisting of Open Software Foundation's Motif widget set and the Xt Intrinsics). The interface provides the user with a convenient tool to construct a meeting request. This include a list of other users from whom to choose meeting attendees (by clicking on boxes, or writing out names). It is also used to notify the user when a mutually acceptable time for a new meeting has been found. The user can choose to reserve the option of a final check before any meeting is scheduled, or may choose to allow his/her agents to schedule meetings without any intervention. We are currently working on extending the interface so that its view can be easily customized.

The interface is also used to input preferences (prefer to have Friday afternoons empty, prefer meetings in the second half of a day, etc.) and priorities (meetings with the boss have higher priority compared to meetings with subordinates, etc.). The **user preferences** module codifies the preferences and priorities of the user which are used as soft constraints to guide the negotiation process. We intend to develop a language by which users can specify their preferences for different types of meetings (depending on the goals of meetings), preference for meetings with other users (a particular user may be more inclined to attend meetings convened by certain users compared to others), preferences for peak density on calendar (a user may choose to attend a seminar if he/she does not have any other lengthy commitment that day). Only some of the functionalities of this module have already been implemented. These include specifying preferences for meeting with different users and preferences for different meeting lengths. For each of the other users in the system, a user can specify his/her preference either by typing in a real number between 0 and 1, or by clicking on a color bar that represents the same continuum. Preferences for meeting lengths are similarly specified. We are currently working on the part of the interface that allows users to specify preferences for meeting in different parts of the week. The basic mechanism is to first select a color from the color bar to represent a preference level, and then click and drag over a part of the calendar to label it with the chosen preference level.

The **negotiation module** implements our theoretical findings (corroborated by experimental simulations), and selects the appropriate calendar access/modification and communication actions necessary to schedule the meetings in the working memory following the preferences of the user. Though this module is the most well-grounded in basic research, we are continuously updating it to accommodate negotiation on meetings with new constraints that we have not studied in our previous research and also to utilize additional forms of user preferences. The **working memory** component simply contains a trace of all communications over meetings that are being currently negotiated. Our implementation is completely object oriented with meetings, proposals, and even users treated as objects. Thus methods are defined to create new meetings in working memory, to add proposals to meetings, and to remove from working memory a meeting that has been scheduled. Objects corresponding to meetings that are scheduled are written out to a history file which can be later consulted for rescheduling or learning models of the scheduling activities of other users. The arrival of meeting request from the user or a bid or a proposal from another scheduling agent triggers the negotiation module to take appropriate actions. The negotiation module uses the **calendar manipulation** module to access/update the current status of the user calendar. It uses system calls to both read and update the status of the user calendar as maintained by the calendar management software. We intend to add to the functionality of this module that will allow users to directly merge schedule changes (which may arise if the user personally scheduled a meeting while away from the computing facilities, e.g., while on a business trip) with the current calendar.

The **message constructor/decoder** is the workhorse in the system which handles the



```
Mail Tool V3: View Message 221
From mahend@aboltabol.mcs.utulsa.edu Mon Sep 26 15:43:14 1994
Date: Mon, 26 Sep 94 13:46:31 -0700
From: mahend@aboltabol.mcs.utulsa.edu (Mahendra Sekaran)
To: sandip@kolkata.mcs.utulsa.edu, hale@euler.mcs.utulsa.edu
Subject: *MS*mahend:26Sep94:13:46:02-HI-P-1
Content-Type: text
Content-Length: 365

Meeting-ID        : mahend:26Sep94:13:46:02
Host              : Mahendra Sekaran
Topic             : Discuss issues on Scheduler
Invitees          : Sandip Sen, John Hale
Proposal          : 10/5/94-14:00, 10/6/94-9:00
Response          : NA
Duration          : 1:30
Priority          : 1
Deadline          : 9/30/94-17:00
Acceptable times  : 10/5/94, 10/6/94 [8:00-12:00]
```

Figure 3: E-mail message containing a proposal for a meeting.

communication between meeting schedulers. Communication between scheduling agents uses the electronic mail system. On being invoked by the negotiation module to send a message, this module constructs a message with a specially formatted subject and body and then calls the mailing software to send out e-mails to the recipient agents. This module also continually polls the system mailbox to check if a meeting scheduling message has arrived (identified by a special header). When it finds such a message, it translates the e-mail message into an appropriate internal format and posts it on the working memory.

Figure 3 shows a snapshot of a mail message announcing a new meeting. The information contained in the header string denotes that this is a meeting scheduling message for a meeting with an ID of "mahend:26Sep94:13:46:02" (formed by appending the login id of the user originating the meeting request with the time at which the request was made). It also specifies that this is a message from the host to an invitee, is a proposal, and is a part of the first round of negotiation on the meeting. The body of the message contains the meeting ID, the user originating the meeting, meeting topic, invitees to the meeting, preferred time intervals being proposed for the meeting (any one of which may be chosen), the response (which would be filled in by the invitee agents when responding to this proposal), expected duration of the meeting, a suggested priority of the meeting (invitees may assign very different priorities to the same meeting), deadline by which a decision on this meeting has to be made, and the set of acceptable time intervals for this meeting (in this case, the meeting can be held either anytime on the 5th of October, or in the morning of the 6th of October). The exact format of the message is evolving as we enhance the capabilities of the other modules of the meeting scheduling system.

Continuing Work

Some of the other extensions that we are actively pursuing include extending the protocol to schedule meetings by choosing representatives from groups (a meeting with a representative from each of the marketing, purchasing, and research division), developing a grammar to specify complex relationships between meetings, creating an explanation facility to justify scheduling decisions to the user, assigning meeting priorities based on pooled and normalized features (which include keywords in meeting descriptors, participants, meeting length, current schedule load, etc.), and learning user preferences by observing users choose from alternative solutions.

The present implementation is being used by a small group of researchers in our department, and we are using the feedback to improve the interface and expand the functionality of the system. Results have been encouraging so far, but we are also aware that the level of expertise of these group of researchers is very different from the average user of such a system. After we have added in some of the functionalities mentioned above, allowing a more structured and easier interaction with the system, we plan to include less sophisticated users to test the system.

On simulation runs, we have been able to scale the system to a large number of users (> 50) without any deterioration in system performance. The implemented system, however, needs more testing before we can use it in such large work groups. The techniques developed so far will work well for meetings with upto approximately 10 users. For scheduling meetings with larger number of attendees, it is unlikely that a common free time can be readily found. The cancellation schemes that we have developed [1] needs to be augmented with mecahnisms to utilize user preferences to handle these meetings with a very large number of attendees. Our system can handle any number of meetings or any meeting densities on the user calendars. The success rate of scheduling meetings without canceling previously scheduled meetings, however, decreases with increased densities on the calendars and when long meetings are being scheduled more frequently. Some of the meeting constraints like scheduling a meeting only if another meeting is scheduled can be easily handled. Other constraints like scheduling a meeting with less number of attendees than specified and/or for a shorter length if necessary, are more difficult to process because of the number of different ways such constraints can be met.

We have done extensive testing in simulation of the performance of our distributed meeting scheduling system with different strategy choices [1]. The number of rounds of negotiation required to schedule a meeting, the percentage of meeting requests that can be accommodated without canceling previously scheduled meetings, and the distribution of meeting density over the calendar varies widely with different strategy choices, and with different number of users, meeting request frequency, etc. We have developed a probabilistic scheme by which agents can automatically choose the scheduling options expected to minimize the number of rounds of negotiation [2]. To schedule typical meetings with 3 to 5 attendees for 2 to 3 hours, our agents take 2 to 3 rounds of negotiation when calendars are 70 to 80% full.

Our work is based on a good understanding and characterization of the approaches needed by automated schedulers to adapt to changing environmental conditions. The applicability of the system will depend largely on how flexible the system is to the demands, constraints, and preferences of the associated user. The basic implementation being in place, we are now concentrating our efforts on developing structured knowledge representation and reasoning mechanism that allows users to easily specify requirements, and allows the automated negotiator to efficiently use those requirements while scheduling meetings.

Acknowledgments

This research has been sponsored, in part, by the National Science Foundation under Coordination Theory and Collaboration Technology grant IRI-9015423 and a Research Initiation Award IRI-9410180, and by a grant from Bellcore. I would like to thank Edmund Durfee for his invaluable suggestions and guidance during much of this work.

References

[1] Sandip Sen. *Predicting Tradeoffs in Contract-Based Distributed Scheduling.* PhD thesis, University of Michigan, December 1993.

[2] Sandip Sen and Edmund H. Durfee. On the design of an adaptive meeting scheduler. In *Proc. of the Tenth IEEE Conference on AI Applications*, pages 40–46, March 1994.

[3] Sandip Sen and Edmund H. Durfee. The role of commitment in cooperative negotiation. *International Journal of Intelligent and Cooperative Information Systems*, 3(1):67–81, 1994.

[4] Thomas W. Malone, Kenneth R. Grant, Franklyn A. Turbak, Stephen A. Brobst, and Michael D. Cohen. Intelligent information-sharing systems. *Communications of the ACM*, 30(5):390–402, 1987.

[5] Communications of the ACM, july 1994, volume 37, number 7, 1994. Special Issue on Intelligent Agents.

[6] Phil Clark. Office automation: Automation gains municipal ground. *American City & County*, page 10, January 1987.

[7] J. F. Kelley and A. Chapanis. How professional persons keep their calendars: Implications for computerization. *Journal of Occupational Psychology*, 55:141–156, 1982.

[8] Irene Greif. PCAL: A personal calendar. Technical Report TM-213, MIT Laboratory for Computer Science, Cambridge, Mass, 1982.

[9] Jonathan Grudin. Social evaluation of the user interface: Who does the work and who gets the benefit? In H. Bullinger and B. Shacketl, editors, *Human Computer Interaction – INTERACT87*, pages 805–811. North Holland, 1987.

[10] Eric Taub. Sharing schedules. *MacUser*, pages 155–162, July 1993.

[11] Lisa Dent, Jesus Boticario, John McDermott, Tom Mitchell, and David Zabowski. A personal learning apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 96–103, July 1992.

[12] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, July 1994.

[13] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.