Satisfying User Preferences While Negotiating Meetings

Sandip Sen, Thomas Haynes, and Neeraj Arora
600 South College Ave.
Department of Mathematical & Computer Sciences
The University of Tulsa
600 South College Avenue,
Tulsa, OK 74104-3189
e-mail:[sandip,haynes,arora]@euler.mcs.utulsa.edu

Abstract

Our research agenda focuses on building software agents that can facilitate and streamline group problem solving in organizations. We are particularly interested in developing intelligent agents that can partially automate routine information processing tasks by representing and reasoning with the preferences and biases of associated users. The distributed meeting scheduler is a collection of agents, responsible for scheduling meetings for their respective users. Users have preferences on when they like to meet, e.g. time of day, day of week, status of other invitees, topic of the meeting, etc. The agent must balance such concerns, proposing and accepting meeting times that satisfy as many of these criteria as possible. For example, a user might prefer not to meet at lunchtime unless the president of the company is hosting the meeting. We apply techniques from voting theory to arrive at consensus choices for meeting times while balancing different preferences.

1 Introduction

Routine information processing needs in organizations can be efficiently automated. We are interested in designing and implementing software agent 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. To effectively act as a surrogate for the user, the associated agent must have some model of the user. The model can either be input by the user or learned from interaction with the user. Along with the model there needs to be a decision making system which applies the model to problem solving scenarios. Effective surrogate agents can greatly facilitate group problem solving by providing new opportunities or enhancing current capabilities of sharing, organizing, and presenting information of mutual interest. Such a class of computational tools, called group support systems or groupware, have been recognized as a key component to enhancing the productivity of cooperative groups in organizations [Greenberg, 1991, Grudin, 1994, Johansen, 1991, Krasner *et al.*, 1991, Rasmus, 1995].

Our work has focused on the problem of how an application domain for intelligent surrogate agents can be analyzed, understood, and represented such that these agents can make appropriate adaptations to their environment, to carry out tasks on behalf of human users. Our particular domain of inquiry has been the meeting-scheduling application, and elsewhere we have analytically developed and experimentally verified quantitative predictions of performance for various strategies for proposing, counter-proposing, and committing to meetings [Sen and Durfee, 1991, Sen, 1993, Sen and Durfee, 1994, Sen and Durfee, 1996]. It did not address many implementation issues like communication medium, user interaction, use of preferences, etc. In this paper, we focus on representing and using user preferences to categorize acceptable/unacceptable meeting proposals, and to generate and rank alternative meeting times to propose. We also provide some details about the current implementation of our meeting scheduling system. The major difference between this implementation and other work on meeting scheduling [Dent *et al.*, 1992, Kautz *et al.*, 1994, Maes, 1994] is that we are more interested in efficiently automating the scheduling process rather than learning about user preferences.

Krasner *et al.* [Krasner *et al.*, 1991] have identified several dimensions along which groupware systems can be evaluated. In the following we rate our automated scheduling system along each of those dimensions:

Human-machine allocation of work: The focus of this work is the automation of most of the tedious and repetitive activity that is performed by humans. Human input is critical for the automated meeting scheduling agent to act as a capable surrogate.

Artifact or process focus: The focus of our application is on producing a schedule preferred by the associated users. So our work aids the group to develop the desirable end-product rather than supporting their negotiation for arriving at the end-product. That is the process of arriving at the end-product is automated, but user preferences regarding the nature of the end-product are honored.

Time: Task interaction can be synchronized or asynchronous. The automated meeting scheduler allows users to process meeting sceduling information asynchronously.

Place: The primary purpose of a distributed meeting scheduler as ours is to allow people to cooperatively arrive at a consensus choice from a geographically distributed setting.

Task: The task being solved by the meeting scheduler is negotiating meeting times. This may involve negotiation which is captured by our preference based contracting protocol.

Group context: The social conventions adopted in our approach is that any person (or his/her agent) can initiate a meeting request. We also assume that others are completely free to accept, reject, modify meeting proposals according to their own preferences. Communication between the schedulers follows a standardized, well-understood protocol.

Group composition: Though a peer group assumption is made regarding connectivity between users, the hierarchical structure of the organization is embedded in encoded user preferences.

In his categorization of groupware tools Johansen [Johansen, 1988] have observed that most groupware system facilitates the users to work at different times in different places. Our work is aimed at building "surrogate" agents that allow users in groupd to perform multiple tasks at the same time by automating some of its information processing needs. We view this line of research to be essential in increasing the productivity of groups in organizations.

In contrast to most of the currently available software for centralized calendar management and meeting scheduling, our approach to 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 satisfying preferences of this individual [Sen, 1997]. 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).

A key characteristic of autonomous information processing agents is that the agents honor

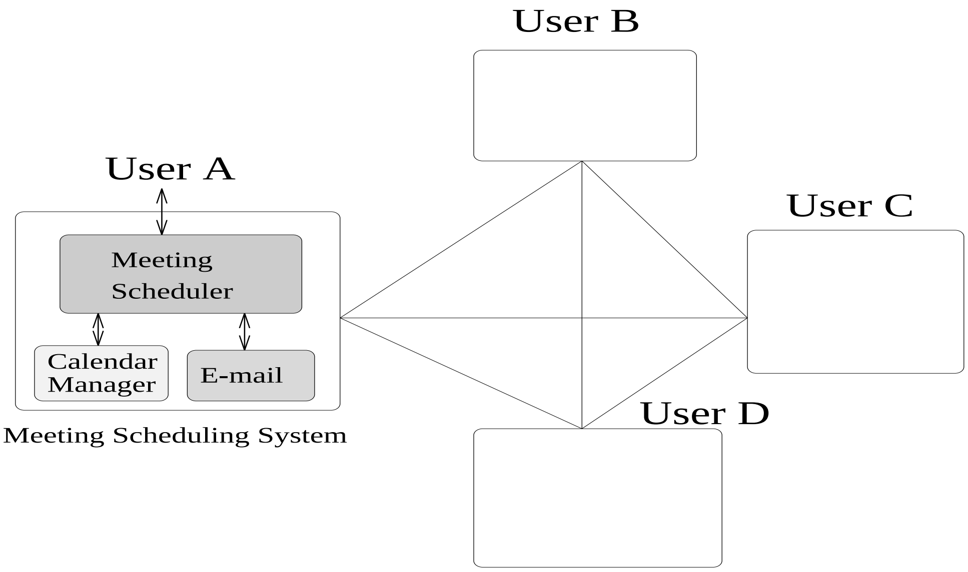

Figure 1: Distributed meeting scheduling over a network of computers.

and follow user preferences. While scheduling meetings, users have preferences for various aspects of the meeting details. As each user might have different preferences for day of week, time of day, host, other invitees, etc., surrogate meeting scheduling agents have to consider these details while negotiating meetings on the behalf of its associated user. But it is often the case that user preferences are mutually conflicting. We use techniques from voting theory to formally represent and reason with conflicting preferences. These techniques provide formal guarantees regarding desirable outcomes of the reasoning processes.

The rest of the paper is organized as follows: Section 2 discusses the role of intelligent software agents in organizations and introduces our agent based meeting scheduling system; Section 3 describes how meeting scheduling agents represent and reason with user preferences; Section 4 presents a series of examples demonstrating the use of voting theory to select preferable options by considering user preferences along a number of dimensions of proposed meetings; Section 5 provides an overview of our current implementation of the automated meeting scheduler; Section 6 presents a summary of the technical contributions of this research and identifies some future directions.

2 Meeting Scheduler

2.1 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 [Malone *et al.*, 1987]. The usefulness of intelligent agents is further appreciated with the advent of the information superhighway [CACM July issue, 1994].

Some of the people in a typical organization spend a large percentage of their working time in scheduling and attending meetings [Clark, 1987, Kelley and Chapanis, 1982]. 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 [Greif, 1982, Malone *et al.*, 1987] in developing automated meeting schedulers have met with limited success, although they are available in a number of office software systems [Grudin, 1987].

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 [Taub, 1993]. 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. Lately, these systems are gaining increased acceptance in business and industry. This is mainly due to peer pressure. But none of these systems satisfy the real need for "intelligent" scheduling that honor user preferences and priorities. This critical requirement of automated scheduling systems is met by our approach to building sophisticated surrogate agents that can store and utilize user preferences and biases.

Similarly, in the AI community, there has been a number of recent projects [Dent *et al.*, 1992, Kautz *et al.*, 1994, Maes, 1994] which assist a human user in scheduling meetings. None of these, however, completely automate the meeting scheduling process. For example, a number of these researchers [Dent *et al.*, 1992, Maes, 1994] have focused on learning personal assistants that learn user preferences by watching the user scheduling meetings. The goal is to assist users in making decisions by suggesting alternatives, and not to automate the process. In the work by Kautz *et al.* [Kautz *et al.*, 1994], the user has to rank each proposed meeting. A similar assumption is made by Ephrati *et al.* [Ephrati *et al.*, 1994] in their attempt to develop a meeting scheduling system which cannot be unduly influenced by a single user. We relieve the user of this cognitive load by automating the ranking of options and finding alternative choices best on stored preferences of the user.

We, however, 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.

2.2 Agent Based 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.

Our scheduling agents use contract-based negotiation [Smith, 1980] 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.

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 them 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).

3 Using user preferences

To be useful, an automated meeting scheduling system has to be adaptive not only to environmental demands but also to user preferences. Whereas our prior work focused primarily on agents adapting to environmental changes [Sen, 1993, Sen and Durfee, 1994], our current efforts are directed towards effectively representing and utilizing user preferences. We realized that for intelligent agent applications to be used by humans, it is essential that these software entities be able to encode and follow the priorities, biases, and preferences of associated users. This realization has forced us to alter the design of the system so that evaluating meetings or alternative proposals for meeting using the user's preference rankings is an integral part of the decision-making of the meeting scheduling agent. In this section, we first give a overview of our proposed voting mechanism to reason with user preferences to schedule meetings in a manner that will satisfy the user. We follow this discussion with a more detailed description of the voting scheme with several examples to illustrate typical scenarios involving tradeoff between conflicting user preferences and how the voting scheme produces a compromise solution.

User preferences for meeting topics, proposed meeting length, host of the meeting, invitees to the meeting, etc. are used to assign a priority for the meeting. For example, user A may be obligated meetings hosted by user B, but not if B was only an invitee. Each user rates all the other users of the system. Similarly he/she provides preferences for meeting topics, meeting lengths, hours of the day, and days of the week, etc. Finally, the user can also weight the preference dimensions, e.g., who is hosting the meeting may be more important than whether the meeting is being held in the morning or in the afternoon. The important point is that every user should be able to specify his/her own particular weightage scheme. Each preference dimension can be examined to determine if the agent believes the user would find the proposed meeting acceptable. Then each preference dimension is given a number of votes in proportion to its weight versus the other preference dimensions. Proposals which gather enough votes are accepted. Details of this voting scheme can be found in Section 3.2.

Preferences are also used to find time intervals to propose during the negotiation phase. By default the system will assume a preferred density profile. The user may, however, specify any alternate profile which, for example, will make the system propose 11AM rather than the default 8AM (or it may propose some time on Wednesday than a time on Monday, etc.).

Using the pooled preferences the scheduler generates a ranking for a proposed meeting, and if this ranking falls below a threshold, it may decide not to attend the meeting. The user has the choice to allow the scheduler to automatically reject such a meeting proposal or a get a confirmation from the user in such cases. Pooled preferences will also be vital for canceling meetings.

We are also careful about limiting the cognitive load imposed on the user of such a system. We provide the user with default preference values for all preference dimensions that are representative of the general trends that are often encountered (e.g., avoid meetings during Friday afternoon, etc.). We anticipate that some users will not take the time and effort to change all the preferences; providing default preferences that are found to e suitable for a wide range of users (e.g., do not like meeting on Friday afternoons, etc.) allows for effective scheduler performance for this class of users. In the future, we plan to investigate learning user preferences, either from observed behavior of the user [Dent *et al.*, 1992, Maes, 1994] (in accepting or rejecting the recommended meeting times) or from querying other agents. This will allow for a better tuning of scheduler performance not only for users who never took the time to set all the preferences, but also for users whose true preferences (as evidenced by their actions) are different from their specified preferences.

3.1 Storing user preferences

We now discuss our choice of representation for storing user preferences. In the following, we use dimensions to refer to user preferences for workloads, constraints like fixed hours of the day, likings for some meeting topics, etc. The term option refers to one of the allowable values for a dimension.

User preferences for meeting topics, proposed meeting length, host of the meeting, invitees to the meeting, etc. are used to assign a priority for an alternative for a meeting. For example, user A may be obligated to attend meetings hosted by user B, but not if B was only an invitee. Each user assigns a value between 0 and 1 for each option of each dimension. For example, in Figure 2, an user has determined his/her preferences for meeting on days of the week. The user also specifies a minimum threshold for that dimension. If the value of the option falls below the minimum criteria, then for that dimension the user would prefer not meeting over meeting. While the options can be mutually exclusive, the sum total of all options under a given dimension need not sum to 1 and can exceed 1.

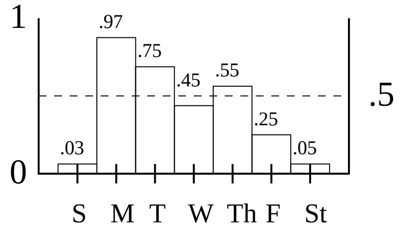

Figure 2: Example preferences for days of the week. The minimum threshold is the dotted line.

Besides rating all of the options for each dimension, the user is also responsible for rating each dimension against each other dimension. Thus some of these options are given greater weight than others, e.g., who is hosting the meeting may be more important than whether the meeting is being held in the morning or in the afternoon. The important point is that every user should be able to specify his/her own particular weight for the options in each preference scheme.

In this research, we have adapted methods developed in the voting theory literature to find compromises between conflicting preferences. Voting is a well understood mechanism for reaching consensus [Ordeshook, 1995, Straffin, 1980]. One of our key goals is that we effectively capture the interactions between the many and conflicting preferences that the user has for a meeting. Voting schemes have been used for ages and several different contexts to arrive at a compromise choice between several candidates. Candidates being voted on can just as well be location of a new natural preserve, political aspirants, one of several science projects to be funded, etc. In general, no one choice appears to be a clear winner along all dimensions under consideration. Voting theory literature provides us with techniques that will select the candidate with some guaranteed properties. With this guarantees, discussed in the next section, the automated scheduler can generate a defensible argument for choosing a particular meeting date when its associated user asks for an explanation. The capability to provide a formal explanation of agent behavior have prompted our use of voting techniques in this work.

We can assign votes to each preference in proportion to their weight against each other. Each meeting proposal can then be voted on by the preferences as to whether the user wishes to meet given the meeting criteria. For each option above the dimension's threshold, yes votes in proportion to the weight of that dimension are recorded, and for each option below the dimension's threshold, no votes in proportion to the weight of that dimension are recorded. In the case that multiple alternatives result in a tie, we can simply choose the earliest scheduled proposal.

Besides using voting by the preferences to decide amongst several alternatives for a meeting, we can also utilize voting for :

- choosing between competing meetings for the same calendar slot,
- deciding whether to attend a particular meeting or not,
- canceling an already accepted meeting in favor of a new proposal,
- counter-proposing a new time for a rejected proposal.

3.2 Reaching Consensus from Preferences

It is often said that human beings often have conflicting preferences. More appropriately, when we have to choose between more than one option, each of which may be more attractive to us than the other for some reason, we have to tradeoff our preferences to arrive at a compromise selection. In the meeting scheduling domain, consider choosing to meet at Monday at 8pm (proposal 1) or Friday at 3pm (proposal 2). Also, suppose the user prefers meeting on Monday rather than Friday and prefer afternoons over mornings. Any choice between proposals 1 and 2 will have to find a compromise between conflicting preferences.

By just looking at ordinal rankings, such a compromise can only be made by flipping a coin. If we however look at the cardinal relationships between the preferences, we can approximate the user's desires. If the user prefers meetings on Monday three times more than meetings on Friday and meetings in the afternoon two times more than those in the morning, the scheduler agent would accept the Monday morning meeting ($V_1 = \frac{17}{24} = 0.71$, $V_2 = \frac{7}{24} = 0.29$). An assumption in this calculation is that the user is indifferent to the choice of weekday versus time of day. If for example, the user also preferred the time of the day four times more than the day of the week, the agent would select the Friday afternoon meeting ($V_1 = \frac{17}{24} * \frac{1}{5} = 0.14$, $V_2 = \frac{70*.8}{24} * \frac{4}{5} = 0.23$).

While such an agent utilizing such a consensus reaching mechanism is not going to always schedule a meeting the way the user wants, it will for the majority of the meetings schedule them the way the user wants if he or she were truthful in selecting the various weights for the preferences. For example, if the user has caught a cold and does not want to meet with anyone at lunch, then the agent is going to be able to capture this one time exception. However, if the user manages to get a lot of customer interaction during informal meetings, lunch might be an ideal time to have meetings. It might even be the case that the user does not want to meet with fellow employees, including the boss, during lunch as it is so important to client relationships.

We now We want an algorithm which will scale well as the number of preferences are increased. The voting scheme we adopt is $O(n^2)$. The chosen methods, discussed below, are guaranteed to provide compromises that have some desirable characteristics. These guarantees, as discussed below, allows us to be confident of the effectiveness of our proposed scheme.

Given a list of alternatives for a given meeting $x_1, x_2, \ldots, x_i$, a set of voters $v_1, v_2, \ldots, v_i$, and a preference list per voter $a_{j1}, a_{j2}, \ldots, a_{ji}$, a pairwise voting scheme calculates the winner of each possible pairing of proposals. In each pairing, the vote proceeds by ignoring the proposals other than those in the pairing. Straffin has listed several criterion to rate the desirability of the outcome different voting rules [Straffin, 1980]:

Pareto If every voter prefers an alternative x to an alternative y, a voting rule should not produce y as a winner.

Condercet Winner If there is an alternative x which could obtain a majority of votes in pairwise contests against every other alternative, a voting rule should choose x as the winner.

Condercet Loser If an alternative y would lose in pairwise majority contests against every other alternative, a voting rule should not choose y as a winner.

Monotonicity If x is a winner under a voting rule, and one or more voters change their preferences in a way favorable to x (without changing the order in which they prefer any other alternatives), then x should still be the winner.

Majority If a majority of voters have an alternative x as their first choice, a voting rule should choose x.

Smith's Generalized Condercet If the alternatives can be partitioned into two sets A and B such that every alternative in A beats every alternative in B in pairwise contests, then the voting rule should not select an alternative in B.

For the meeting scheduler, we believe that the Pareto and Monotonicity criterion are of paramount importance: a weak alternative y should not be selected and improvement of a winning alternative should not cause it to lose. If a proposed meeting P_j is desirable over all other meetings, $P_i, i \neq j$, then none of the other meetings should be accepted by the agent on behalf of the user. Likewise, suppose the user decides to change his or her preferences such that afternoon meetings are now more important than morning meetings. If P_1 had been accepted earlier and was already in the afternoon, then no other meeting should be accepted over it.

With just these two criteria, we would select Borda's voting rule: assign points to an alternative based on its position in a voter's preference list. The last place alternative gets 0 points, the second to last 1 point, and so on until the first place which gets n points. If a voter is indifferent to 2 or more alternatives, then each one is assigned the average of the alternatives. For example, if a voter has the preference list of $p = \{a, \{b, c, d\}, e, \{f, g\}\}$, then the points are awarded as follows: g and f each get 0.5; e gets 2; d, c, and b each get 4; and, a gets 6. The alternative that receives the highest number of votes from all voters is the Borda count winner.

However, the order in which alternatives are considered should not effect the final outcome. If three meetings are proposed, the selection of the agenda should not matter. For example, the order of comparisons $\{P_1, P_2, P_3\}$ should produce the same winning alternative as the presentation of any permutation of the proposals, e.g., $\{P_3, P_1, P_2\}$. This additional criteria causes us to strengthen the Pareto criterion to the Condercet Winner criterion. Based on this additional requirement, we have chosen to implement the Black voting rule. This rule is designed to strengthen the Borda count voting rule: if there is a Condercet Winner, then choose it, else apply the Borda count voting rule.

Voting systems have previously been applied to distributed artificial intelligence domains [Ephrati and Rosenschein, 1991, Rosenschein, 1995, Ephrati and Rosenschein, 1996] and meeting schedulers in particular [Ephrati *et al.*, 1994], but not in the context of cooperative voters. Ephrati *et al.*'s approach focuses on voting by all users of a meeting to reach consensus on an acceptable time for the meeting and is a non–cooperative voting scenario where voters can vote insincerely [Ephrati *et al.*, 1994]. They can therefore use only those voting strategies that cannot be sabotaged by strategic manipulation [Ordeshook, 1995, Straffin, 1980]. However our approach is for different preferences of the same user to vote on what is a preferable meeting time, and hence is a cooperative voting system. Thus even though the Borda count is vulnerable to strategic manipulation [Brams and Fishburn, 1991], we can exploit the cooperative nature of the meeting scheduler agent to avoid manipulation.

Another difference is that they require a cardinal utility evaluation of alternatives while we require an ordinal evaluation. The major problem with their approach is that they require each user to rate all possible meeting times for a meeting, which is time consuming and unlikely to be accepted by ordinary users. We, on the other hand, are proposing to automate the ranking of alternative proposals using stored user preferences, which circumvents the problem in their scheme. Though we have not done this yet, it is possible to combine these two proposals, i.e., each meeting scheduling agent can first rank alternate proposals for a meeting by using a cooperative voting scheme using stored user preferences, and then all the agents involve in scheduling a meeting may use a voting scheme that discourages manipulation to select a mutually agreeable time to schedule the meeting.

4 Example applications of voting schemes

In this section we present several examples to illustrate how the preferences over different dimensions can vote to reach a consensus about scheduling meetings. In Figure 3 we present eight different dimensions relating to preferences for a user Pat for scheduling meetings. In Figure 4, we also present Pat's rankings of the different options for each of the dimensions. To calculate the votes for each dimension from Pat's preference list of all the dimensions, first the ratio of the weight for that dimension to the sum of the weights is calculated, and then the ratios are multiplied by a constant so that all votes are integer numbers. For example, we can calculate the votes for dimension A as:

$$V_A = \frac{P_A * 1000}{\sum_{i=A}^{H} P_i} = \frac{.90 * 1000}{4.92} = 183$$

The resultant votes for each of the dimensions for Pat's preferences is as follows: A:183, B:197, C:163, D:71, E:60, F:142, G:31, and H:153. Unless explicitly stated otherwise, for the rest of the paper, we assume that Pat's calendar indicates that no other meetings have been scheduled in any of the considered slots.

ID	A	B	C	D	E	F	G	H
1	W	9	Boss	N	Y	Mike	Sales	45

Table 1: Single alternative for first meeting to be scheduled by Pat's agent.

The first example we present is when a single meeting request is considered by Pat's agent. The agent is tasked with scheduling alternative id 1, see Table 1. This scenario is considered to be an election between accepting and rejecting the proposal. As discussed in

Section 3.1, an option for a dimension receives yes votes in proportion to the weight of the dimension if the value for that option lies above the minimum threshold line (if it lies on the minimum threshold, there is a tie.). For example, since the proposed day, Wednesday, falls below the minimum threshold ($0.45 < 0.50$), 183 votes are allocated to rejection from the **A** dimension. Summing up all the votes we see that accept gets 491 votes and reject gets 509 votes. Since there are only two alternatives, and reject dominates accept, it is the Condercet Winner, and hence the proposed meeting is rejected. While we do not consider counter-proposals in this example, by just changing the proposal from Wednesday to Thursday, the agent would accept this new proposal.

ID	A	B	C	D	E	F	G	H
2	W	9	VP	N	N	Dan, Sue	Sales	15
3	W	10	VP	N	Y	Dan, Sue	Sales	15

Table 2: Alternatives for second meeting to be scheduled by Pat's agent.

The second example we consider is when two alternatives for a meeting are given to the agent(see Table 2): alternative ids 2 and 3. Before the agent can decide which alternative to select, it must first decide which are acceptable. Again, we hold an election between the alternatives and rejection. Figure 5 represents the pairwise voting between the alternatives. Both alternatives are acceptable as they both beat reject (R). Furthermore Alternative 2 is the Condercet Winner and the agent informs the host agent that it accepts alternative 2. An interesting point, revealed in this example, is that the dimensions which have been given greater weight, by the user in his or her preferences, are responsible for getting a meeting accepted. We can call such dimensions the major dimensions and the dimensions the user does care about as much can be called minor. To decide between multiple viable alternatives both the minor dimensions and differences between option values of major dimensions in the alternative proposals are important. Also, a preference that is major, i.e. important, to one user might be minor to another.

ID	A	B	C	D	E	F	G	H
4	Th	3	Sue	N	Y	Boss, Mike	Development	45
5	Th	12	Sue	Y	N	Boss, Mike	Development	20
6	F	7	Sue	N	Y	Boss, Mike	Development	60

Table 3: Alternatives for third meeting to be scheduled by Pat's agent.

In our third example we again consider a scenario where multiple alternatives for a given meeting are proposed (see Table 3): alternative ids 4, 5, and 6. The pairwise voting is shown in Figure 6. There is no Condercet Winner, but reject (R) is a Condercet Loser. The agent must now carry out a Borda count election of the alternatives, which are ranked in order in Table 4. The calculations for the votes of the different alternatives are as follows:

$$V_4 = 2.5(183) + 3(197) + 2(163) + 0.5(71) + 2.5(60) + 2(142) + 2(31) + 0(153) = 1906$$

$$V_5 = 2.5(183) + 0(197) + 2(163) + 3(71) + 0(60) + 2(142) + 2(31) + 2(153) = 1648.5$$

$$V_6 = 0(183) + 2(197) + 2(163) + 0.5(71) + 2.5(60) + 2(142) + 2(31) + 3(153) = 1710.5$$

$$V_R = 1(183) + 1(197) + 0(163) + 2(71) + 1(60) + 0(142) + 0(31) + 1(153) = 735$$

Alternative 4 wins the election via the Borda count voting scheme and the agent would notify the host of accepting this meeting proposal.

5 Implementation

We are in the process of implementing a distributed meeting scheduling system on a workstation based computing environment, that utilizes the above-mentioned findings. The pro-

Dimension				
A	4,5	R	6	
B	4	6	R	5
C	4,5,6	R		
D	5	R	4,6	
E	4,6	R	5	
F	4,5,6	R		
G	4,5,6	R		
H	6	5	R	4

Table 4: Rankings of election of proposals 4, 5, 6, and Reject (R).

gramming language used for the current implementation is Java [Flanagan, 1996], which was chosen for its portability and object-oriented features.

The architecture of our meeting scheduling agents is as described in Figure 7. 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 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 scheduling system, and uses the user preferences when 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 developing a Java-based GUI for user-agent interaction. Interface functionality include allowing the user to construct a meeting request (See Figure 8). The user may input the topic and location of the meeting requested. A slide bar allows the selection of the length of the meeting (in half hour intervals). The window provides a list of known correspondents. Clicking on a name in the list transfers the name to a bottom slot, and this user can be included in the invitee list of the proposed meeting by using the add button. New users can also be added by typing in their names in the bottom slot and using the add button. User can be deleted from the invitee list by clicking on them on the invitee list window on left and then clicking the delete button. Acceptable times for the meeting can be specified by clicking and dragging the mouse over regions of the calendar on the right. The scheduler automatically shows the user which of the specified times are already occupied by other scheduled meetings. We have also included the option for requesting daily, monthly, weekly, or monthly repeating meetings. Help, cancel, and OK buttons are provided to get more help on filling in the required information, to clear all entered information, and to submit the meeting request.

The interface is 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. The interface is also used to input hard constraints like the working hours, preferences (prefer to have Fridays lightly loaded, 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. For each preference dimension, the user provides rankings between 0 and 1 for each of the options. The user also provides a threshold value for every preference dimension; a given proposal's value for this particular preference dimension has to be greater than this threshold for a 'yes' vote to be casted for this preference dimension (e.g., if two proposals for a meeting is proposed, one for Monday and one for Friday, and the preference rating for these two days are 0.7 and 0.4 respectively with the threshold for days of the week set at 0.5, then a 'yes' and a 'no' vote will be cast for the Monday and Friday proposals respectively from the "Days of the week" preferences). Default values are given for the preferences and the user may modify these values. We further reduce user interaction by requiring users not to rate each acquaintance individually but simply categorizing them as superiors, peers, subordinates in the company or as customers or suppliers outside the company and then attaching preferences to these classes as a whole. The user can attach different weights to different preference dimensions which are used to rate proposals by a weighted voting scheme.

The **negotiation module** implements our theoretical findings, 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 continually updating it to accommodate negotiation on meetings with new constraints 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 a meeting request from the user or a bid or a proposal from another scheduling agent triggers the negotiation module. 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 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 stored calendar.

The **message constructor/decoder** is the workhorse in the system which handles the communication between scheduling agents using 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-mail 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. To prevent cluttering the mailbox of users, we are using the Internet Message Access Protocol, version 4 (IMAP4 [Crispin, 1994]), to monitor and delete messages relevant to the meeting schedulers.

Figure 9 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 "*MS*aroraFriNov1:13:08:12CST1996" (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), expected duration of the meeting, a suggested priority of the meeting (invitees may assign very different priorities to the same meeting), the 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 4th of November, or in the morning of the 5th of November). The format of the message is evolving as we enhance the capabilities of the meeting scheduling system.

6 Conclusions

We have developed an agent-based meeting scheduling system that can automate the task of scheduling meetings between groups of users. By effectively storing and utilizing user preferences, these agents not only relieve the user from routine and repetitive chores, but also promises to reduce the time taken to schedule meetings and produce more effective schedules for organizations because of a more systematic and extensive search of the calendar space. This means that groups of users can now schedule meetings with much less effort and be able to identify meeting opportunities which may be overlooked by a less systematic, manual process of using e-mails, telephone, etc. because of cognitive constraints and limitations. The resultant facilitation of group activities is the precise goal of groupware systems. Our particular application, is particularly noteworthy, because it adds another dimension to the usefulness or applicability of groupware systems.

In this paper, we have presented a voting scheme which allows an agent to arrive at a consensus choice for meeting times while balancing different user preferences. We model the different dimensions of user preferences as voters and conduct elections between different alternative proposals for a meeting "inside" the agent. Because we need not worry about strategic manipulation in this context, we can use computationally efficient voting schemes that also guarantees desirable outcomes. We have designed the system so as to minimize the amount of supervision the user must provide. For example, though the system allows the user to input preferences for a number of dimensions, it can work with default preferences if any user decides not some or all of these preferences.

Some of the 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, 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 this 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.

Acknowledgments:
The authors would like to thank Ed Durfee for his support and suggestions that have had a significant impact on this work. This work has been supported, in part, by an NSF grant IRI-9410180.

References

[Brams and Fishburn, 1991] Steven J. Brams and Peter C. Fishburn. Alternative voting systems. In L. Sandy Maisel, editor, *Political Parties and Elections in the United States: An Encyclopedia*, volume 1, pages 23–31. Garland, New York, 1991.

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

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

[Crispin, 1994] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4. Request for Comments (Experimental) RFC 1730, Internet Engineering Task Force, December 1994.

[Dent *et al.*, 1992] 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.

[Ephrati and Rosenschein, 1991] Eithan Ephrati and Jeffrey S. Rosenschein. The clarke tax as a consensus mechanism amoung automated agents. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 173–178, Anaheim, CA, July 1991.

[Ephrati and Rosenschein, 1996] Eithan Ephrati and Jeffrey S. Rosenschein. Deriving consensus in multiagent systems. *Artificial Intelligence*, 87(1-2):21–74, November 1996.

[Ephrati *et al.*, 1994] Eithan Ephrati, Gilad Zlotkin, and Jeffrey S. Rosenschein. A non-manipulable meeting scheduling system. In *13th International Workshop on Distributed Artificial Intelligence*, July 1994.

[Flanagan, 1996] David Flanagan. *Java in Nutshell.* O'Reilly, 1996.

[Greenberg, 1991] S. Greenberg. Computer-supported cooperative work and groupware: and introduction to the special issue. *International Journal of man-machine studies*, 34(2):133, February 1991.

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

[Grudin, 1987] 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.

[Grudin, 1994] Jonathan Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 35(1):92–105, January 1994.

[Johansen, 1988] Robert Johansen. *Groupware: Computer Support for Business Teams.* Free Press, New York, NY, 1988.

[Johansen, 1991] Robert Johansen. Groupware: Future directions and wild cards. *Journal of Organizational Computing*, 1(2):229, 1991.

[Kautz *et al.*, 1994] Henry Kautz, Bart Selman, Michael Coen, Steven Ketchpel, and Chris Ramming. An experiment in the design of software agents. In *National Conference on Artificial Intelligence*, pages 438–443, 1994.

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

[Krasner *et al.*, 1991] Herb Krasner, John McInroy, and Diane B. Walz. Groupware research and technology issues with application to software process management. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):704–712, July/August 1991.

[Maes, 1994] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[Malone *et al.*, 1987] 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.

[Ordeshook, 1995] Peter C. Ordeshook. *Game Theory and Political Theory: An Introduction*. Cambridge University Press, 1995.

[Rasmus, 1995] Daniel Rasmus. Groupware: Reconnecting with human intelligence. *PC AI*, 9(6):27, 1995.

[Rosenschein, 1995] Jeffrey S. Rosenschein. Multiagent planning as a social process: Voting, privacy, and manipulation. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, page 431, San Francisco, CA, 1995. MIT Press. (invited speaker talk).

[Sen and Durfee, 1991] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling: Preliminary results. In *Proceedings of the ACM Conference on Organizational Computing Systems*, pages 55–68, November 1991.

[Sen and Durfee, 1994] 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.

[Sen and Durfee, 1996] Sandip Sen and Edmund H. Durfee. A contracting model for flexible distributed scheduling. *Annals of Operations Research*, 65:195–222, 1996.

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

[Sen, 1997] Sandip Sen. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the First International Conference on Autonomous Agents*, pages 308–315, February 1997.

[Smith, 1980] 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.

[Straffin, 1980] Philip D. Jr. Straffin. *Topics in the theory of voting*. The UMAP expository monograph series. Birkhauser, Boston, MA, 1980.

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

A: day of week Which day of the week is the meeting to be held.

B: time of day Which hour of the day is the meeting to be held. Note we can subdivide the hours into a sufficiently fine granularity, i.e., 30, 15, or 5 minute intervals. Also, while the user might have different preferences per day of the week, for our examples, we assume they utilize the same preferences for each day.

C: host Who is hosting the meeting? The user's boss, the VP, the president, or Bob down the hall. Notice that the user's preference on the host should also incorporate the priority associated by the host to this particular meeting. Perhaps the user almost always attends meetings proposed by his/her boss, unless the boss associates a low priority to the meeting.

D: lunch If the meeting is at lunch time, will lunch be provided or is the user expected to brown-bag?

E: in building Will the user have to leave the office building. Note this could be extended to in city, in state, in country, and overseas.

F: invitees Who are the other users invited to the meeting. Since the combinations of users can explode, some scheme must be present to dynamically determine the associated ranking for this preference. For our examples, we simply use the weighted average of all invitees, excluding the host and the user.

G: division Which division of the company is the meeting going to be held? This could also include the topic of the meeting.

H: length of meeting How long is the meeting supposed to last?

Figure 3: Dimensions for scheduling a meeting for Pat.

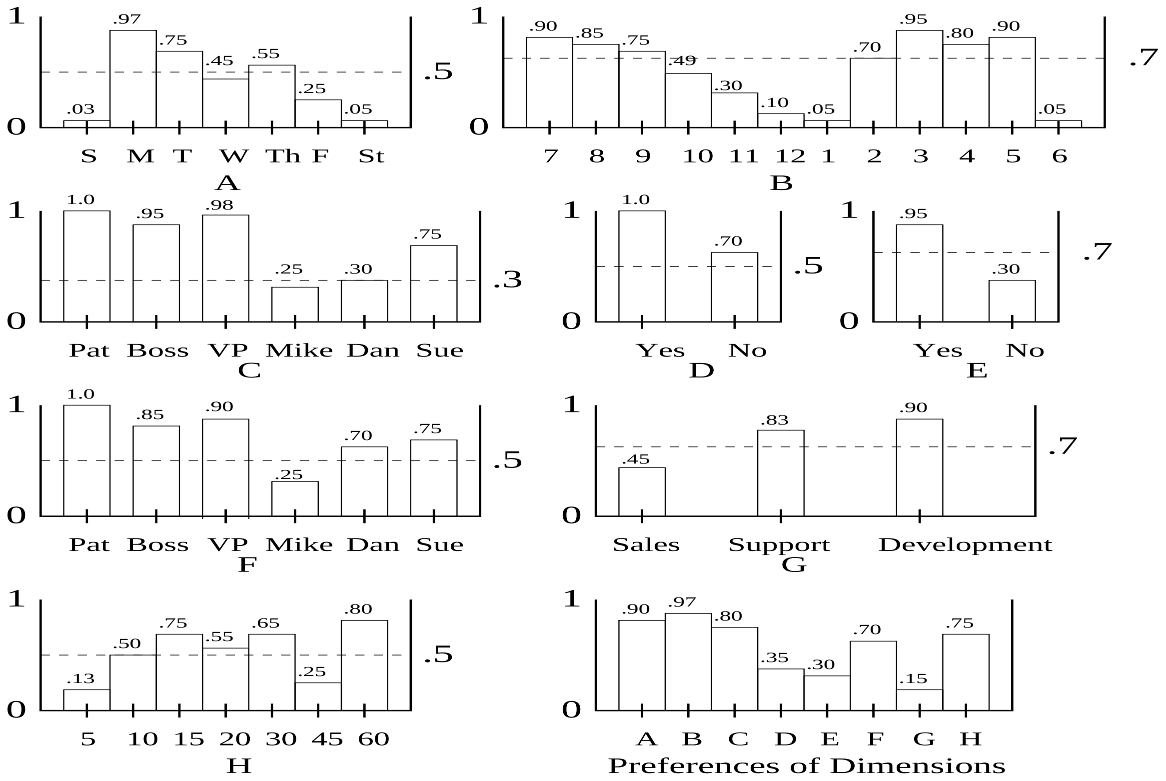

Figure 4: Example dimensions for user Pat. The minimum thresholds are the dotted lines. Each dimension is labeled to the corresponding dimension in Figure 3. Lastly, we also present Pat's preference list of the dimensions.

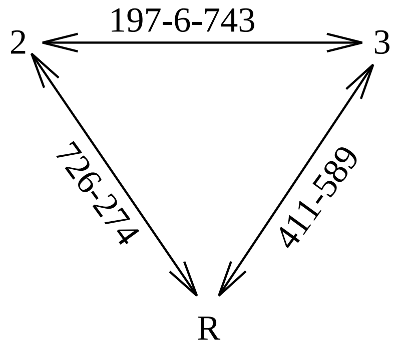

Figure 5: Pairwise voting between alternatives 2, 3, and reject (R). For the pairwise voting, the results are given as Win-Lose-Tie. If there is not a tie, this information is left out.

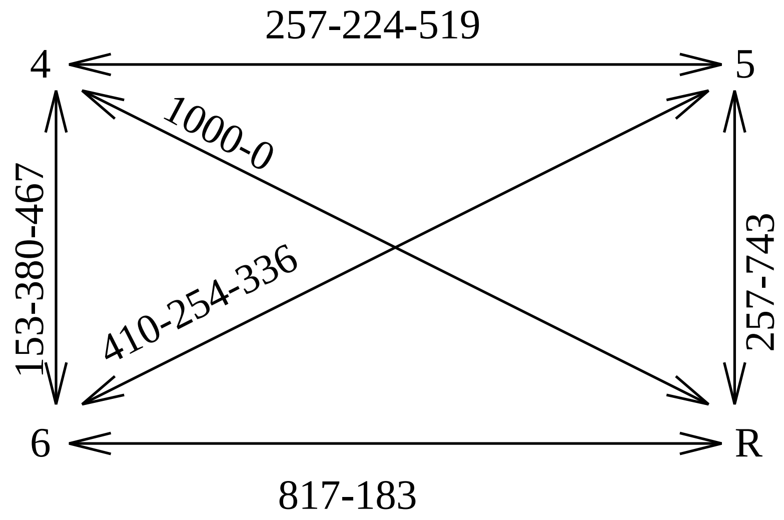

Figure 6: Pairwise voting between alternatives 4, 5, 6, and reject (R). For the pairwise voting, the results are given as Win-Lose-Tie. If there is not a tie, this information is left out.

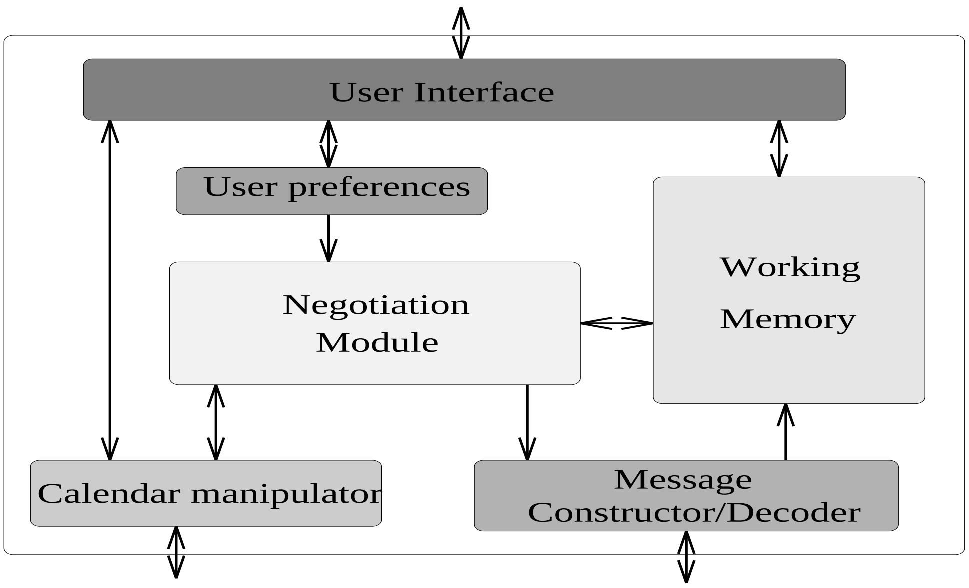

Figure 7: The architecture of the automated meeting scheduler.

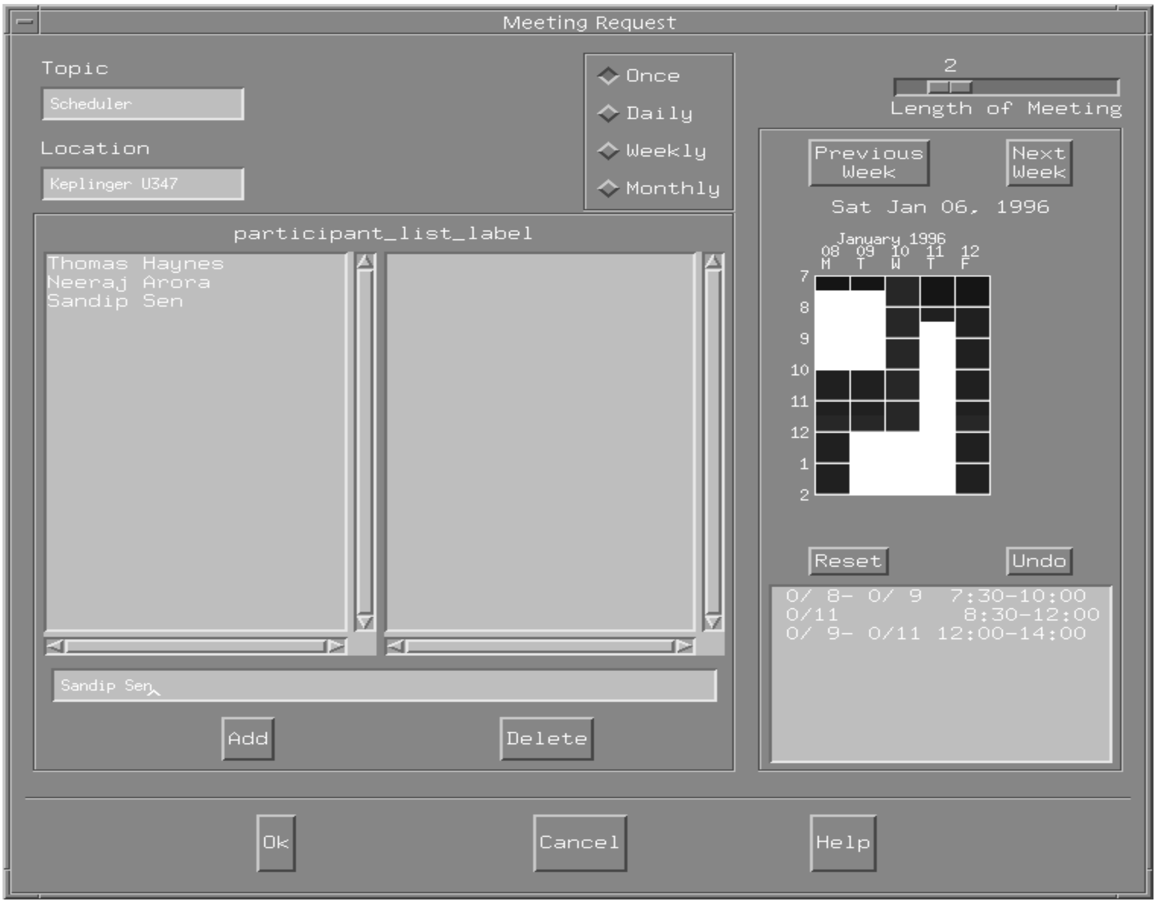

Figure 8: The interface window for requesting meetings.

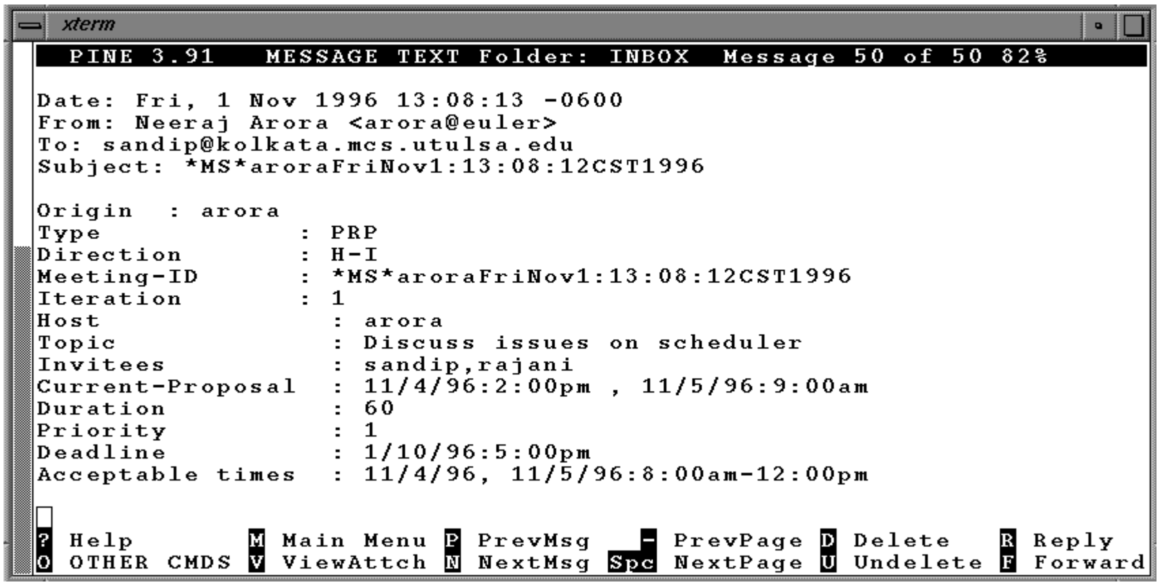

Figure 9: E-mail message containing a proposal for a meeting.