A distributed multi-agent meeting scheduler

Elhadi Shakshuki ∗, Hsiang-Hwa Koo, Darcy Benoit, Daniel Silver

Jodrey School of Computer Science, Acadia University, Wolfville, Nova Scotia, Canada B4P 2R6
Received 15 June 2006; received in revised form 31 October 2006
Available online 25 April 2007

Abstract

Electronic calendars are important tools that are used by consumers on a daily basis. However, scheduling a meeting that involves persons with different commitments and preferences remains a difficult task. Meeting scheduling is difficult because current calendaring applications cannot handle the responsibility of automatically and autonomously managing time slots. This paper presents a distributed multi-agent system architecture in which each person is represented by an agent. These agents automatically and autonomously work together to assist different users to book meetings on their behalf. Each agent has the capability to manage, negotiate and schedule tasks, meetings, events, appointments for its assigned user. In this multi-agent system, the agents coordinate their activities and negotiate on behalf of their associated users to find a solution that satisfies the users’ meeting requirements and preferences. A prototype of this system is implemented to demonstrate how the agents can automatically book meetings.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Software agents; Scheduling; Negotiation; Calendars

1. Introduction

Computing technology has grown rapidly over the past several decades. Many hardware components and software applications have been developed to help humans solve their living needs. Networking applications such as Internet browsers, email clients, web calendars, etc., and office tools such as word processors, calendars, schedulers, spreadsheets, etc., are examples of popular services and applications. A scheduling system or calendar is a typical and popular personal tool that supports people’s daily routines. Many people use different methods and tools to notify them of upcoming meetings.

For example, Microsoft Outlook [7] is a popular email and scheduler application. There are many other scheduler applications, including ones that run on different operating systems, such as Novell Evolution [9] on the Linux [5] platform. Scheduler applications have two types of formats: single and distributed. In single scheduler applications, users are allowed to manage their reserved meetings without any interaction or awareness about other users’ schedulers. In distributed meeting scheduler applications, users are able to utilize a network and reserve meetings on other
people’s schedulers through a middleware application. For example, the combination of Microsoft Outlook [7] and Microsoft Exchange Server [6] allows a user to book meetings on another user’s scheduler. Figure 1 shows an example of booking meetings through a network. This allows one user to view the schedule of and book meetings with other users without direct interaction with those users. Thus, a user can access another user’s scheduler through the Internet and book a meeting, thus utilizing free time slots. A Personal Digital Assistant (PDA) [10] is another very popular computing device. It is small, thin, lightweight and easy to carry. It includes a few useful and necessary programs such as address books and a calendar. A scheduler application allows the user to schedule their meetings and appointments in their PDA.

Although the aforementioned schedulers are useful tools, they are unable to work on behalf of their users to handle the responsibility of booking meetings automatically. The user is required to manually book meetings. If there is a conflict with the desired meeting slot, then resolving the conflict is also the responsibility of the user. Resolving conflicts usually results in having to directly contact the users involved in the meeting.

One promising solution for this scheduling problem is to use software agents to help users to automatically manage their meetings with minimum conflicts. This paper describes a multi-agent system that acts as a mediator between the users and the scheduling environment. The agent for each user is provided with the meeting’s constraints and the preferences of the user. The main objective of each scheduler agent is to find an acceptable time slot that best fits the constraints and preferences for its user. When conflicts occur, the agents involved are responsible for proposing an alternative solution that may require overriding the preferences of some users. Overriding the preferences of some
users may result in the movement of reserved time slots or scheduling a meeting through an otherwise blocked-off time (such as lunch). The agents will attempt to complete this task with minimal preference overrides and minimal interaction from the users.

An overview of a meeting scheduling environment is shown in Fig. 2. This environment consists of users who are interested in booking meetings, agents that work on behalf of their users, and calendars that are used by the agents to manage meetings. These agents are called Meeting Scheduler Agents (MSAs) and are able to send meeting requests to and receive responses from other agents. These requests are made through the network to manage meetings on behalf of the users. These agents are also able to reschedule meetings whenever it becomes necessary. The communication between agents is established using two methods: direct message exchange and email. The scheduler agents provide users with an interface to input meeting requests with the desired date and time, and then update the calendar to display the meeting. When agents begin to schedule a meeting, they send the meeting request with the date and time to the agents of the other participants in the meeting. If the request can be fulfilled, then the meeting is booked for everyone involved. If the request cannot be fulfilled, then a conflict has arisen. As a result, a new time must be chosen or other meetings for some participants must be shifted. The main function of the agents is to find a time slot that best fits all meeting participants. In a situation where an agent needs to negotiate with other agents, it is equipped with several negotiation strategies. These negotiation strategies are built based on the constraints and preferences for the user, allowing the agent to reschedule meetings in different time slots while still within the preferences of the user.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes our proposed system architecture and explains its components. Negotiation strategies are discussed in Section 4, followed by the implementation details in Section 5. Section 6 provides experimental results to demonstrate the functionality of the proposed system. Finally, Section 7 provides the conclusions and future work.
2. Related work

Recently, many researchers have considered agent-based technology as a proposed design paradigm and used this technology to develop new systems to act on behalf of their users. Much attention has been given to the development of scheduling systems, but little work has been accomplished in automating these scheduling systems. Most attempts to automate schedulers lack a description of the proposed system architecture, have an incomplete implementation, or possess limitations in the proposed negotiation strategy when a conflict occurs.

In the CMRadar project [1], a personal assistant agent architecture has been proposed for scheduling meetings. The system architecture consists of an extractor, a manager, a scheduler, and an interface to display calendar data. This system is partially implemented and integrated with Microsoft Outlook. Of the portion implemented, it is not clear how some components are implemented, nor is it clear how the system is integrated with Microsoft Outlook. Researchers in [1] have also proposed three useful negotiation strategies including greedy, bumping and NCost. In these strategies, it is assumed that the initiator always proposes a single time slot and a receiver either accepts or rejects the time slot. If a rejection occurs, the initiator proposes a new time slot.

A similar approach has been proposed by Berres and Oliveira in [11] to provide a distributed meeting scheduler using an agent-based approach. In [11], two negotiation strategies are proposed to find free time slots for booking meetings, including “first possible” and “take best” strategies. In this approach, when an agent sends a request for a meeting to other agents, the agent has to wait for all responses from all invited agents, leaving no clear limit to terminate the negotiation process. Agents are implemented using Java Agent Template Light (JATLite) [3], but no description of the agent architecture is given.

Shintani et al. presented a method for multi-agent negotiation when implementing a distributed meeting scheduler [14]. They proposed a negotiation mechanism based on the multi attribute utility theory to provide effective consensus. The main focus of this work is the negotiation, which is only one aspect of meeting scheduler systems. Along similar research lines, Sen had previously proposed an automated distributed meetings scheduler using agents [12]. The main focus of this work was the negotiation strategies proposed. The work included a brief description of the system architecture and implementation.

Most of these approaches use email as the primary communication mechanism when scheduling meetings. We consider this approach to be inefficient for several reasons. Firstly, as the number of meeting invitees increases, so does the number of email messages. If one attendee cannot attend a meeting, then the rescheduling process will involve large numbers of emails. This is something that we wish to avoid. Secondly, using email as a method of communication results in a meeting scheduler that can only work on behalf of the user when the user is accessing their email. If a user is offline and not accessing their email, then it is not possible for the system to access the emails needed for communication. This method will also cause problems for attendees who use different types of email systems (POP3 vs. IMAP) and different email clients (Outlook vs. Eudora). Finally, many email systems include intelligent spam filters to reduce email load on already overextended email servers. Direct communication can avoid problems with these systems.

The main objective of this paper is to develop an automated meeting scheduler based on an agent system architecture that is able to help users book meetings according to their meeting priorities. The main design principles of the system are to address some limitations that exist in current meeting scheduler systems. TCP/IP sockets are used as the primary choice of communication for inter-agent messages and communication with the server. Direct communication will avoid the problem of email messages being mistakenly flagged as spam. Sockets will also speed up the scheduling process, as overworked email servers can often take a number of hours to deliver email messages. Email is used as a secondary communication tool when TCP/IP sockets are not possible. Email is also used to notify users of meetings or inform users of the status of the server. We propose that agents use a voting strategy to overcome unsatisfactory solutions proposed by existing negotiation strategies. The voting strategy will eliminate unnecessary wait time and maintain control of the negotiation process. Users interact with the system through a graphical user interface and are allowed to submit requests and impose time and preference constraints on meetings.

3. System overview and architecture

The architecture of the proposed system is designed to help users book meetings automatically based on their interests and preferences. This system is client-server based and consists of a Meeting Scheduler Server (MSS) and
a Meeting Scheduler Agent (MSA), both shown in Fig. 3. The meeting scheduler agent contains the following components: calendar displayer, manager, learning processor, scheduler, email generator, and an internal database.

The main function of the calendar displayer component is to provide the graphical interfaces the user needs to view scheduled meetings. The manager component is responsible for negotiating with other agents when a conflict occurs. All user decisions are monitored and written to a database by the learning processor component for use in future interactions. The scheduler component is responsible for finding free time slots and sending them to the manager component. The function of the email generator component is to generate notification messages and send them to other users. All the required information such as time slots and dates are saved in the internal database.

The meeting scheduler server contains the following five components: communication component, knowledge base component, message generator component, message extractor component, and backup database component. The communication component is responsible for establishing a link between agents and facilitates protocols for sending, receiving and interpreting all messages. The backup database component is responsible for updating all the required information in case of a hard-drive crash or malfunction. The message extractor component is responsible for parsing incoming messages and formatting the outgoing messages to the required message formats. The message generator component displays the messages in a readable format to the user. The knowledge base component stores all the required information including the previous negotiation cases that are used by the learning processor.

4. Negotiation

Negotiation is a form of decision-making where two or more parties jointly search a space of possible solutions with the goal of reaching a consensus [2]. One of the main goals of the proposed agent is to negotiate with other agents.
when there is a conflict. To achieve this goal, the agent is equipped with three negotiation strategies, namely first come first served (FCFS), high rank (HR), and voting strategy (VS): The main objective of each negotiation strategy is to find an acceptable time slot that meets the agents’ preferences assigned by the users.

In our proposed agent system architecture, the negotiation strategies are built inside the meeting scheduler manager. When the user interacts with its agent, the user enters the required meeting information and preferences, including date, time, subject, location, description, user priorities for meeting participants and meeting mobility (fixed or movable). The agent utilizes this information to find time slots that best fits the desired preferences. The following explains some of the basic terms used in our proposed negotiations strategies:

- **Host agent**: a host agent is the agent who requests a meeting. Every meeting can have only one host agent.
- **Guest agent**: a guest agent is the agent that receives a request for a meeting. A meeting must have at least one guest agent.
- **Fixed and movable meeting slots**: A fixed meeting means that the agent is not allowed to move this slot during negotiation. Fixed slots are represented by a zero. A movable meeting means that the agent has the ability to shift a reserved meeting slot to an alternative one, and is represented by a value of one. Fixed and movable meeting slots include all preferences such as date and time.
- **User priority**: this term is used to assign an integer value to reflect the users’ meeting priority. Each user is allowed to assign different priorities to users on his or her contact list. Priority values are used to determine the priority of the meeting. Values are assigned from one to ten, where ten is the highest priority. Meetings are rescheduled during conflicts based on the priority value given to meeting attendees by the person who is initiating the meeting.
- **Highest user priority**: when a conflict occurs, the guest agent looks at the user priority set by the host agent for that guest agent. This information is automatically extracted from the host agent’s contact list. The guest agent then considers the previously scheduled conflicting meeting. If the guest agent booked the previously scheduled meeting, then the guest agent looks at its own user priority of the meeting attendees. If the guest agent did not book the previously scheduled meeting, then the guest agent considers the user priority that host agent has for it. The appropriate user priority is then compared with the host agent priority of the guest agent. The guest agent determines, based on the user priorities, which meeting will be rescheduled. It should be noted that when attendees are added to a meeting, the host is informed of the highest user priority of the meeting attendees. The host is given the option to override the highest user priority value and assigned a new value to the meeting. This allows the user to either increase or decrease the priority of a given meeting.
- **Negotiation cycle**: is the total time required to send a message from the host agent to guest agents and the time required to receive the responses sent by the guest agents.
- **Shifting meeting process**: this process allows an agent to send a shift meeting message to all attendees to a new proposed available time slot, whenever the requested meeting time slot is reserved. The agent has a threshold for extra waiting time to allow late rescheduling meeting messages to arrive.
- **Negotiation time**: it is the maximum time allowed by the agents to reach an agreement. The negotiation time ($N_t$) is calculated using:

$$N_t = W_{tsp} - kW_{tsp},$$

where $W_{tsp}$ is the waiting time before starting the shifting meeting process and $k$ is a threshold that can have a value from 0 to 100.

### 4.1. First come first served strategy

When an agent applies this first come first served (FCFS) strategy, it accepts incoming meeting requests in order. For example, when a host agent sends requests for a meeting to guest agents, the guest agents will automatically accept if they have empty slots during the proposed meeting time. This strategy is utilized when the requested time slots are available. If the requested meeting time conflicts with guest agents’ local schedulers, the high rank (HR) strategy is employed to book the meeting. The algorithm for this strategy is shown in Listing 1.
When a meeting request is received, then:
– Get status of time slots from local scheduler
If status of time slots = “available”
– Accept requested meeting
Else
– Use HR strategy

Listing 1. FCFS algorithm.

If FCFS fails Then
Get status of conflicted meeting: fixed or movable meeting
If status = “movable” then /* Conflicted meeting is movable */
– Determine which meeting has the higher user priority (conflicted or requested)
If “requested” has higher user priority then
– Add the requested meeting
– Start the shifting meeting process for the conflicted meeting
Else
– Perform the rescheduling process for requested meeting
End /* Conflicted meeting is fixed */
– Perform the rescheduling process for the requested meeting
If conflicts cannot be resolved or negotiation time expires Then
– use voting strategy

Listing 2. HR algorithm.

4.2. High rank strategy

The high rank (HR) strategy is applied when agents are unable to use FCFS to book a meeting. HR is only applied when the agent has one meeting booked and a meeting request for the same time period. When the agent applies HR, it first checks to see if the booked meeting is fixed or movable. If the booked meeting is fixed, then the agent has to propose a different time slot for the proposed meeting. If the booked meeting is movable, then the agent must determine if the proposed meeting is of higher priority than the booked meeting. The agent will check the users’ priorities to decide who has the higher meeting priority. This strategy allows the agent with the “high rank” to have their preferred time slot. For example, assume that agent A has booked a meeting with agent B. Agent C then requests a meeting with agent A in the same time slot. Agent A will then consider its rank of agent B and the rank it has received from agent C. The agent will assign the meeting time to the agent with the higher rank, allowing the lower ranked agent to negotiate for a different meeting slot. In this example, if the priority ranking of agent A from agent C is higher than agent A’s ranking of agent B, then the meeting between A and C will occur in the conflicted time slot and the meeting between A and B will be rescheduled. Likewise, if agent A’s ranking of B is higher than agent C’s ranking of A, then the meeting between A and B will not be moved and agents A and C will have to negotiate a new time. This algorithm is intended to make sure that the meeting with the higher priority participants is given priority over meeting times in the user’s schedule. The algorithm for HR strategy is shown in Listing 2.

4.3. Voting strategy

The voting strategy (VS) is the last resort for the agents to find a meeting slot when other strategies fail. This strategy is utilized when agents cannot reach an agreement during the negotiation time limit. The server is assigned responsibility of this strategy to ensure a fair decision. Before discussing more details about how the VS is implemented, we will first explain the format used to determine meeting times. In our system, each 15 minute time period is considered a time slot. Each time slot is numbered consecutively from the beginning of the day until the end of the day. If a one hour meeting is required, then four consecutive time slots will be grouped together. That group will then be used when communicating with other agents concerning a meeting time.

In the case where negotiations fail and a meeting time cannot be agreed upon, the VS is used. The VS consists of the host agent finding as many non-overlapping groups as possible, during the timeframe when the meeting can occur, and sending them to the server. Once the server receives all of the groups, it sends them to the agents of the meeting participants. The agent for each meeting participant will then “vote” on each group, informing the server if
If HR fails Then
– Host agent: sends all possible blocks to the server
– Server: sends all possible blocks to all guest agents
– Guest agents:
  Assign yes or no message for each block
  Respond to Server
– Server:
  Waits for predefined time to receive the responses from guest agents
  Pick the block that has majority yes
  If more than one block has the same majority Then
    ◦ Pick the block based on highest user priorities
      If more than one block has the same high priority Then
        Pick a block based on the preferences of the host (morning, afternoon, etc).
        If the user preferences do not exist or do not apply Then
        Pick the block randomly
Listing 3. Voting strategy.

they are available for that meeting time. Once all of the agents respond to the server (or some predetermined timeout occurs), the server will pick the group with the most votes and hold the meeting in that time slot. If there are multiple groups that have the same number of votes, then the server will choose the group where the meeting has the highest priority.

Meeting priority is determined by adding all of the priorities of the meeting attendees. If the host has chosen to override the highest user priority for a given meeting, then attendee priorities will be increased or decreased appropriately. If the value of the override is lower than the highest user priority, then attendee priorities higher than the override will be reduced to the value of the override. If the value of the override is higher than the highest user priority, then all attendee priorities will be increased to the value of the override. These increases and decreases in user priority will only apply to the meeting in question.

In the case where multiple groups have the highest user priority, then the slot will be chosen based on the preferences of the host agent, such as a morning time slot if the host prefers morning meetings. If the host agent has no preferences or the preferences do not apply to the decision at hand, the slot will be chosen randomly from the group. Once a new time has been selected for the meeting, the server will inform all of the agents about the meeting. Agents will also be informed that this meeting is non-negotiable. Agents will then be responsible for overriding user preferences or rescheduling other meetings in order to make this meeting possible for their users. The user will have the option to override the agent and skip the meeting, or cancel other meetings in order to make the meeting in question.

The algorithm for voting strategy is shown in Listing 3.

4.4. Agents modes of operations

It is possible for the meeting scheduling client to operate with agents in two different modes – online and offline. In online mode a host agent and a guest agent are used. The host agent initiates contact with the guest agents to schedule a meeting. The host agent can use FCFS and HR strategies to make sure that meetings have valid time slots, collect information about meetings that need to be shifted, or propose new time slots for meetings under negotiation. Since the guest agent does not initiate meetings, it must first wait for a request to come from the host agent before it will perform any actions. Once a request has been received, the guest agent will use the same FCFS and HR strategies to handle the requested meeting.

Offline mode allows a single agent to perform FCFS and HR strategies in its local scheduler. With offline mode, all messages are kept locally at the host agent until the agent is back online. Once the agent is back online, all of the messages generated will be sent to the appropriate agents and server. In some cases, the agent will generate an email for another user. When in offline mode, this email will also be queued and sent when the agent is back online. Offline agents are not able to utilize the voting strategy.

5. Implementation

A prototype of the proposed system is implemented using JAVA [4]. This system is implemented using the client-server architecture. The server is implemented as middleware that connects all agents. The server function-
ality includes identifying users’ authorizations, formulating outgoing messages and parsing incoming messages. All messages exchanged between agents are formulated in a format similar to Knowledge Query Manipulation Language (KQML) [13]. The agents communicate with each other using TCP/IP stream sockets. A database is used at the server to store meeting requests for agents that are offline. The server will inform agents of buffered meeting requests when the agent comes back online. The server also uses the database to create a back-up copy of the client’s preferences and to maintain the agent directory service. The database is implemented using the MySQL [8] database management system.

The system allows users to book meeting while either online or offline. In online mode, TCP/IP is used to communicate with the server and other agents. When the user selects offline mode, the user can schedule meetings without connecting to the server. Each agent uses a MySQL database to store local information such as meeting times and other agents’ addresses.

The interaction between the agent and the user is accomplished with graphic user interfaces (GUI). One example of user interaction with the agent through a GUI is shown in Fig. 4. In this example, the offline user is communicating with the agent to book a meeting. The send button is deactivated, only allowing the user to save the meeting to the local scheduler.

The user is informed of meetings in their local schedule by way of a GUI. The interface supplies needed meeting information to the user, including start time, end time, date, location, subject and the user’s priorities for the meeting. An example is shown in Fig. 5.

Our implementation uses time slots of 15 minutes. This creates 96 equal time slots in each day. Each time slot is assigned a number, allowing meeting start and end times to be represented by time slot numbers, as shown in Fig. 6. The 15 minute time slot indicated by the meeting start and end numbers are included in the meeting time. For example, for a meeting from 8:00 to 10:00, the starting time slot is 33 (8:00–8:15) and the end time slot is 40 (9:45–10:00), meaning that time slots are inclusive in the meeting. Each meeting is assigned a Meeting ID (MID).

The user’s schedule is stored in a local database which is used by the client’s local computer. When a meeting is reserved in the schedule, the meeting ID is inserted in the appropriate time slot in the user’s local scheduler. A default value of $-1$ is assigned to any time slot that does not have a meeting scheduled.

Fig. 4. A user books a meeting while in offline mode.
6. Results

To demonstrate and assess the feasibility of our proposed meeting scheduler system, some examples were constructed and tested. These examples demonstrate how the agents interact with one another to handle requested meetings. In these examples, the system is implemented with four agents, one for each user. Table 1 shows the user’s contact lists and priorities. The priority of each contact is different for different users because the importance of a contact differs for each user.

6.1. Example 1

This example is used to evaluate the agent’s ability to book a meeting in its own local scheduler. In this example, Alex interacts with his agent and requests a meeting with the following information. The graphical user interface shown in Fig. 7 is used.

- Subject: Clean the apartment
- Location: Home
- Time: 10:00–11:00
- Date: July 25, 2006
As soon as Alex submits the meeting to the agent, the manager component receives the information from the interface. The manager component then queries the scheduler component to determine if the meeting time is available. If the meeting time is available, the scheduler will provide this information to the manager. If the meeting time is not available, then the manager will be informed and the manager will be required to propose a new meeting time. Once an available meeting time has been agreed upon between the manager and scheduler components, the manager will then determine if other users are invited to the meeting. Seeing that no other users are invited to the meeting, the manager will then inform the scheduler to book the meeting and update the calendar display.
In this example, Alex requested a meeting from 10:00 to 11:00. Once the manager and scheduler components have agreed that this time slot is available, the scheduler component will update the calendar display. The adjusted timeline is shown in Fig. 8.

6.2. Example 2

This example demonstrates the agent’s ability to move a previously booked meeting when a conflict occurs with a higher priority meeting. Assume that Alex has already reserved the meeting time slot as shown in Example 1. Abdur wishes to book a meeting with Alex and HsiangHwa. Abdur interacts with his agent and submits a meeting request which includes the following information. The graphical user interface shown in Fig. 9 is used.

- Subject: Invitation to attend thesis defence
- Location: CAR 113
- Time: 9:30–11:30
Abdur submits his meeting information to his agent through the agent’s user interface. The manager component queries the scheduler to determine if the meeting slot is available. Finding that the meeting slot is available, the manager component will then check if other users have been invited to the meeting. Finding that Abdur has requested that Alex and HsiangHwa attend the meeting, the manager component formulates a request message (shown in Listing 4) and sends it to Alex’s agent and HsiangHwa’s agent.

When HsiangHwa’s agent and Alex’s agent receive the meeting proposal, they will begin by parsing the message for meeting information and action commands. The actions of the agents will depend on the action command received. HsiangHwa does not have any meetings booked for this time period, so when HsiangHwa’s manager component requests a meeting from his scheduler component, the time slot is free and the meeting is easily booked. HsiangHwa’s agent will then confirm acceptance of Abdur’s requested meeting with a confirmation message.

Alex’s agent performs the same steps as HsiangHwa’s agent, but Alex’s scheduler will inform his manager that there is a scheduling conflict. This conflict is because of the meeting that Alex has scheduled from 10:00 to 11:00 on the same day. The conflict is shown in Fig. 10.

In order to resolve the time conflict in Alex’s schedule, the manager component of Alex’s agent compares the priority of the two meetings. The first step is to determine if the first meeting is “fixed”. If the first meeting is fixed and cannot be moved, the agent will respond to Abdur’s agent with a proposed new time slot. If the first meeting is not fixed and can be moved, Alex’s agent will employ the ranking strategy to determine if the initial meeting should be moved. If the initial meeting has higher priority attendees, then Alex’s agent will inform the other agents of a conflict and will propose a new meeting time. In our example, Alex’s initial meeting has lower priority attendees. Before attempting to resolve this scheduling conflict, the manager component in Alex’s agent will wait for a pre-determined amount of time. This waiting period is introduced due to the possibility that other invited agents will have a conflict with the proposed meeting time. If another agent has a conflict and proposes a new meeting time, Alex’s agent will not need to reschedule his initial meeting. If no conflicts occur and the waiting time expires, Alex’s agent will attempt to reschedule the initial meeting. The rescheduling process includes sending messages to attendees to inform them of Alex’s conflict and to begin negotiating a new meeting time. If the initial meeting cannot be rescheduled, the agent will inform attendees of the initial meeting that they cannot attend in order to allow the higher priority meeting to take place. In this case the initial meeting is rescheduled from 8:30 to 9:30 and the proposed meeting is added to Alex’s
local scheduler. Figure 11 is a graphical user interface to show that the new meeting time has been updated in Alex’s schedule. In the case where a meeting is rescheduled, a message is formulated to send to meeting attendees to move the meeting, and an email message will be generated to inform the users of the change, too. Listing 5 is an example of the shifting meeting message format. In this particular case, Alex’s appointment didn’t have other attendees. Thus, there is no need to notify other agents.
Listing 5. Message content to shift a meeting.

```plaintext
(shift_meeting
  (host_by (name host_nickname) (user_ID host_user_ID) (meeting_ID ##))
  (subject "Enter a subject")
  (location "Place")
  (meeting_time (from ##:##) (to ##:##))
  (date "MM DD, YYYY")
  (invitees
    (guest
      (name guest_nickname)
      (user_ID guest_user_ID)
      (priority ##))
    (fixed_or_movable_meeting ##)
    (highest_user_priority ##)
  )
)
```

Fig. 12. Rescheduling the meeting is required.
6.3. Example 3

This example is used to evaluate the agent’s ability to negotiate meetings where conflicts dictate that not all invitees will be able to attend. Consider the situation in Example 2. HsiangHwa, Alex and Abdur all have a meeting scheduled between 9:30 and 11:30, all with a priority of seven. Assume that Elhadi requests a meeting with these three people from 10:00 to 11:00, with a priority of seven. When each of the agents receives the meeting request, each agent will compare the priority of the new request to the priority of the previously scheduled meeting. Since the priorities are equal, each agent will respond back to all of the agents involved with a message indicating that they have a conflict. All messages will include new proposed meeting times. Figure 12 shows rescheduling the meeting is required.

The new proposed meeting times are then considered in the order that they arrive at the host agent. If one of these time slots is appropriate, then the meeting is booked in that slot. A graphical user interface to show the update of the rescheduling meeting time is shown in Fig. 13.

Otherwise, the negotiation cycle continues until the user-defined negotiation time expires. At that time, the host requests that the server use the voting strategy to resolve this conflict. The voting strategy involves the host agent sending all possible meeting slots to the server. An example of this message from the host agent is shown in Listing 6.

The server will then send all possible meeting slots to each participant in the meeting. An example for the server sending this message to all guest agents is shown in Listing 7. Each participant will then vote “yes” or “no” to each proposed meeting slot and return that information to the server. An example for guest agents sending the returning message that includes all results is shown in Listing 8. For our example, let us assume that the host agent (Elhadi’s agent) has two free time slots in the afternoon: one from 13:00 to 14:00 and another from 16:00 to 17:00. These two time slots are sent to the server by Elhadi’s agent. Once the server receives these time slots and the attendee list, the server will contact each attendee and ask them to vote on the proposed time slots.

```
(request_voting
  (host_by (name Elhadi) (user_ID 4) (meeting_ID 3))
  (subject “Multi-Agent System Presentation”)
  (location “CAR 207”)’’
  (possible_blocks
    (available (from 13:00) (to 14:00))
    (available (from 16:00) (to 17:00)))
  (date “July 25, 2006”)
  (invitees
    (guest (name Alex) (user_ID 7) (priority 6))
    (guest (name HsiangHwa) (user_ID 6) (priority 7))
    (guest (name Abdur) (user_ID 5) (priority 6)))
  (fixed_or_movable_meeting 1)
  (highest_user_priority 7)
)
```

Listing 6. An example of a message for requiring the voting strategy.
Listing 7. Example voting request message during the voting strategy.

Listing 8. Example response message during the voting strategy.

Assume that HsiangHwa’s agent replies that he is available for both time slots. Alex’s agent replies that he is only available for the 13:00 to 14:00 time slot, and Abdur’s agent replies that he is only available for the 16:00 to 17:00 time slot. The schedules are shown in Fig. 14. Once the server has received all responses or waited the specified period of time to receive responses, the server will determine the best meeting time based on number of participants and participant priority. The time slot with the maximum number of participants is chosen as the meeting time. If multiple slots have the same maximum number of participants, then we must use attendee priorities. Using the priorities from Table 1, we can see that Elhadi has ranked HsiangHwa at seven, Alex at six and Abdur at six. The first proposed meeting slot would have three participants (Elhadi, HsiangHwa and Alex) while the second proposed meeting slot would also have three participants (Elhadi, HsiangHwa and Abdur).

Because both proposed meetings have the same number of participants, we will determine the best meeting time by ranking the time slots based on attendee priority. This is done by adding the priorities for each attendee to generate a value. The highest value shows the best time slot. The host is not included in these calculations as they can attend all meetings under consideration. For the first meeting slot, the attendee priorities are seven (HsiangHwa) and six (Alex), giving a total of 13. For the second meeting slot, the priorities are also seven (HsiangHwa) and six (Abdur), giving a total of 13. Since neither time slot has a higher calculated value, the server must then choose from the two time slots. The server will take into consideration the preferences of the host and make a decision from there. In this case, if the host prefers late afternoon meetings, then the 16:00 to 17:00 time slot will be chosen. If the host prefers early
afternoon meetings, then the 13:00 to 14:00 slot will be chosen. If the host has no preference, then the slot will be chosen randomly.

7. Conclusions and future work

This paper proposed an automated meeting scheduler architecture that consists of several agents that act as representatives for their users to manage, negotiate and schedule tasks, meetings, events, and appointments. A prototype of this system was implemented using Java. The system was tested on several example scheduling cases. The negotiation strategies we proposed were successful in allowing the agents to resolve meeting conflicts and book meetings on behalf of their owners.

Our future work will include the incorporation of machine learning techniques in the learning component of the agent. This will allow the agent to monitor the behaviour of both the user and other agents to predict the users’ preferences. We plan to allow the user to have more control in adjusting the priority of the meeting. In addition to allowing the user more control over meeting priority, we will investigate the feasibility of allowing the agent to learn and predict meeting priorities based on attendees, the topic and the other meetings in the user’s schedule. Another area of research is the creation of an automated database agent that will be responsible for storage and backup of all agent data. This database agent will also be responsible for pushing updates to the local database at each agent. We plan to test our system for scalability to determine the impact of the proposed negotiation strategies on scalability. We will further develop our negotiation strategies to allow a host agent to specify a full block of free time in which to book a meeting during the voting strategy. Our new negotiation strategy will include the ability to use a “best fit” algorithm to book the meeting with the optimal number of attendees.

References