Visualizing and Manipulating Complex Calendar Scheduling Information

Andrew Faulring* and Brad A. Myers†

Human-Computer Interaction Institute
Carnegie Mellon University

ABSTRACT

Calendar scheduling is a difficult task for people who have overbooked calendars and many constraints. We are collaborating with artificial intelligence researchers, who are developing an intelligent calendar scheduling agent that gathers availability constraints, searches for times that satisfy the constraints, and negotiates with invitees when no satisfactory time is found for the constraints. The agent will never be able to act with complete autonomy because, just like a good human assistant, it will need to consult its supervisor when a task is under-specified, has ambiguous instructions, deviates from the normal, or when the task has changed. Furthermore, users need an interface to tell the agent all of their constraints, to see the agent's proposed solutions, and to respond to the agent's questions. Currently, calendar applications do not allow users to specify scheduling constraints such as how preferable a free time is for scheduling a new meeting or to what extent an existing meeting can be rescheduled. These requirements inspired the "availability bar," an interaction and visualization technique for complex, multi-dimensional calendar scheduling constraints. Availability bars were specifically designed to be embedded in calendar applications. Additionally, availability bars will help people who schedule their calendar by hand.

CR Categories and Subject Descriptors: H.5.2 [Information Interfaces and Presentation]: User Interfaces---Interaction styles, Natural language, Graphical user interfaces (GUI); H.4.1 [Information Systems Applications]: Office Automation---Time management; I.3.6 [Computer Graphics]: Methodology and Techniques---Interaction techniques.

Additional Keywords: Constraints, visualization techniques, rich human-agent interaction, interface agents, intelligent user interfaces, calendars, calendar scheduling.

1 INTRODUCTION

People perform many problem-solving tasks, such as scheduling meetings, arranging travel, and making purchasing decisions, that require searching for solutions that satisfy *complex constraints*. The constraints in these problems describe personal preferences for each property of the solution and the relative priority of each constraint. Many of these tasks also involve constraints from multiple people and the relative priority of each person's constraints. The problem-solving task involves an iterative process of: (1) collecting all constraints, (2) evaluating each possible option against the constraints, and (3) adjusting the constraints if either no solution or too many solutions satisfy the constraints. The process continues until the person in charge accepts a solution or decides that no solution exists.

We are investigating constraint specification and visualization in the context of calendar scheduling, a common example of a constraint-based problem-solving task. A person trying to organize a meeting ("the meeting organizer") may employ a variety of electronic tools to help with this task including individual calendar applications, groupware (shared) calendaring systems, email, spreadsheets, and web accessible databases of conference room availability.

Existing shared and non-shared calendar applications do not represent complex availability preferences for busy professionals such as managers and university faculty, making it difficult for meeting organizers to solve calendar scheduling tasks. While shared tools support collecting each invitee's free and busy times, they do not allow invitees to specify implicit, individual preferences such as how early in the day they are willing to meet, when they prefer to eat lunch, and so forth. Palen's ethnographic study of scheduling at a large company with a shared calendaring system observed that a user's calendar may not include all of their actual time commitments, such as meeting preparation and commute time [16]. Palen notes that a meeting organizer browsing the shared calendars of invitees in search of common available times must "assess the quality of what appears to be free time in someone's calendar," an assessment that relies on constraints and preferences not explicitly represented in the shared calendar system. Hence Palen comments that scheduling is more of a "satisficing" rather than "optimizing" task.

Despite their benefits, shared tools are not always available since they usually require a centrally managed server, for example a Microsoft Exchange Server. Some organizations choose not to operate such servers, and those that do generally limit access to their members, rendering the tools useless when scheduling meeting with people outside of the organization. Falling back to non-shared calendar tools forces the meeting organizer to collect availability constraints, generally via email, from each invitee. An invitee receiving such a request must look at their calendar to find times when they are available and then manually convert the calendar's graphical representation into prose that describes either explicit ("Monday: 10–12") or vague ("Wednesday afternoon") constraints and preferences. Upon receiving responses back from the invitees,

* faulring@cs.cmu.edu
† bam@cs.cmu.edu

Submitted for review to InfoVIS 2006
—
Do not cite

the meeting organizer converts the natural language into a representation better suited for finding a meeting time and place. Both conversion processes are tedious and error prone.

Neither type of calendar system supports other important subtasks. First, when no common free time exists, the meeting organizer might try to negotiate with the invitees to determine if any of them can adjust their availability, perhaps by rescheduling a meeting. Calendaring tools provide no help for deciding with whom to negotiate nor do they support the actual negotiation process. Second, given a set of possible meeting times, the meeting organizer has no simple method to evaluate how well each option satisfies the complex constraints. Options that appear equally good when just considering availability constraints may differ drastically when each invitee's preference for each time is considered. Third, the meeting organizer may need to adjust the relative importance of each constraint. For example, some invitees' availability constraints may carry a stronger weight because their attendance is more important or they are less likely to reschedule their calendar to accommodate a new meeting. The meeting organizer may want to organize the constraints into a hierarchy in which all the constraints in one level must be satisfied before constraints in the next level are considered. For example, faculty availability constraints might be placed in a higher level above their students' constraints.[1] Finally, just keeping track of all the constraints and the status of different negotiations can be quite a challenge for the meeting organizer, especially when multiple meetings are being scheduled concurrently.

2 Calendar Scheduling Agents

We are investigating user interfaces as part of Radar (*R*eflective *A*gent with *D*istributed *A*daptive *R*easoning), a large-scale interdisciplinary project that is building a suite of intelligent agents to help office workers more quickly complete routine tasks. Other researchers are building agents such as CMRadar for calendar management [13, 15], a Webmaster [19], and a Space Planner.

CMRadar uses multi-agent negotiation strategies and schedule optimization algorithms based upon user preferences to help the user by: (1) negotiating meeting times on the user's behalf, (2) finding possible schedules given external meeting requests, and (3) asking the user which schedules they prefer. Other areas where a calendar scheduling agent might help include collecting information and constraints from the web or other people, asking the user to supply unknown data that the agent believes would be most useful, or telling the user the constraints that are blocking any possible solutions.

Consider the range of actions that an agent might take. At one extreme, the agent might recognize an opportunity to help, but take no action. At the other extreme, the agent might take action, perhaps just reporting the results after the fact. Horvitz developed a computational framework that could aid an agent in making the decision to "do nothing," "ask the user," or "take action" [8]. Isbell and Pierce extended Horvitz's work by defining a continuum of agent proactivity between these two extremes [9]. In the vast majority of cases, the agent will need to interact with the user, and so visualization and interaction techniques are needed to allow the user to successfully interact with an intelligent calendaring system. This should come as no surprise as people spend much time interacting with other people to resolve conflicts and negotiate options.

In addition, the techniques that we will describe in this paper would also be helpful for people doing calendar scheduling tasks by hand. Our approach is to put the user's interaction with the agent within the context of the user's existing direct manipulation applications such as the Microsoft Outlook calendar tool or the Palm Desktop. This approach contrasts with prior work where the user and agent engaged in a chat-like conversation in a separate window [17].

3 Design Scenarios

We present novel designs of visualization and interaction techniques to help manage the complexities of calendar scheduling. We present a series of scenarios showing how our designs could be used. In the first scenario, a user has received a request to attend a meeting. We present the "availability bar" visualization and interaction technique, which allows this user to easily respond by specifying availability preferences for regions of time in their calendar. In the second scenario, a user wants to schedule a large multi-person meeting. We extended the availability bar technique to show each invitee's availability in the context of a calendar. For any meeting time under consideration, the design visualizes how well the time satisfies the constraints and highlights any constraints that are violated. In the third scenario, a user receives a request for a meeting that might require rescheduling existing meetings. Our design shows how an agent might present multiple alternate schedules. Finally, we describe visualizations of unknown values for a meeting's start, end, and duration properties that allow a user to put a meeting in their calendar without implicitly committing it to a particular time or duration.

4 The Availability Bar

We describe the "availability bar" visualization and interaction technique, which helps a meeting invitee respond to a meeting request. The technique helps an invitee to specify their availability constraints. We developed the availability bar visualization to address many of the problems with calendar applications described above.

An availability bar visualizes a user's availability for new meetings over the course of a day. Many factors can affect a user's availability, including the user's preferences and the reschedulability of existing meetings. The availability bar breaks the day down into regions of time, each of which is assigned a preference level drawn from continuum of values ranging between "required" and "unacceptable" (see Figure 1). The "required" level indicates that only the specified time is allowed, which is only meaningful in response to a specific request for a meeting. The "unacceptable" level indicates that the specified time is not allowed.

4.1 Preference Levels

Graphically, the continuum is represented by a gradient of a single hue as seen in Figure 1.1. The "required" extreme is assigned a very saturated value of the hue, which should draw attention to it. The "unacceptable" extreme is assigned white

[1] Or, the hierarchical ordering might be reversed when dealing with students, who cannot reschedule their classes.

(completely unsaturated). For the intermediate values, saturation decreases proportionally with decreasing preferability levels. The labels may be adapted to particular situations. For example, a constraint defining how reschedulable a meeting is could use labels such as: must be rescheduled, more reschedulable, reschedulable, neutral, least reschedulable, and not reschedulable.

Figure 1.2 shows an earlier design for the preference level visualization, which used a multi-hue gradient: green ("required") to gray ("neutral") to red ("unacceptable"). When a user is looking for good times to schedule a meeting, red draws attention to the unacceptable times, which is not helpful, and it becomes more difficult to scan for areas of darker green (the best times). Also, in black-and-white, or for someone who is color-blind, both ends of the continuum look the same.

4.2 Embedding Availability Bars

Figure 2.1 shows a user's calendar with an availability bar on the right of the day (*a*), which encodes the user's availability at different times of the day. The less saturated ("less preferable") regions before 9:30 (*b*) and after 5:00 (*k*) represent the user's preference to not have meetings early in the morning or late in the afternoon. The four unscheduled regions between 9:30 and 3:30 (*c*, *e*, *g*, and *i*) represent "more preferable" times for scheduling a meeting. The meeting with Nicholas is important and cannot be easily rescheduled so the availability bar has the "unavailable" preference level (*d*). The meeting with Ashley can be rescheduled if necessary, so the availability bar has the "less preferable" preference level (*h*). Lunch time is assigned the "neutral" preference level indicating that the user could meet during that time if necessary (*f*). The preference level can vary over a region of time. For example, the preference level for the region from 3:30 to 5:00 decreases continuously from "more preferable" to "less preferable," indicating a decreasing preference for meetings during that time (*j*).

The user can manually create, resize, and delete regions of the availability bar, and change the preference level for each region. We envision several mechanisms for setting the availability bar that would relieve the user of the need to do so for each day. The user may specify default availability preferences on a daily, weekly, or monthly basis that are automatically applied to each day. An intelligent agent could also learn the user's general time of day preferences, which it might set as the daily default [15]. The calendaring system could also use rules to update the availability bar as meetings were added, removed, or deleted. For example, a rule might set the availability during meeting with one's boss to "unavailable." An intelligent agent might create rules based upon learned knowledge of the extent to which an existing meeting can be rescheduled [14].

4.3 Painting Availability

In a variety of situations, a user will need to specify availability preferences: when responding to an email or to an agent, in a shared calendar system, as a global preference, when modifying an availability bar, or in general whenever availability information is needed. We developed the "painting availability" technique as demonstrated in Figure 2.2–3 to make it easy for a user to express availability preferences.

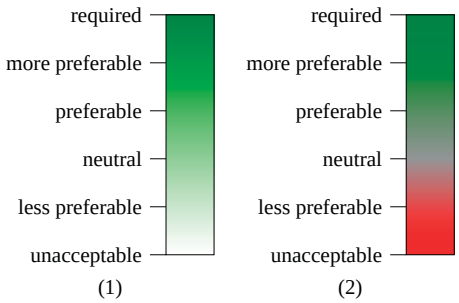

Figure 1. (1): A gradient of a single hue encodes the preference level continuum. (2): An earlier design with a multi-hue gradient.

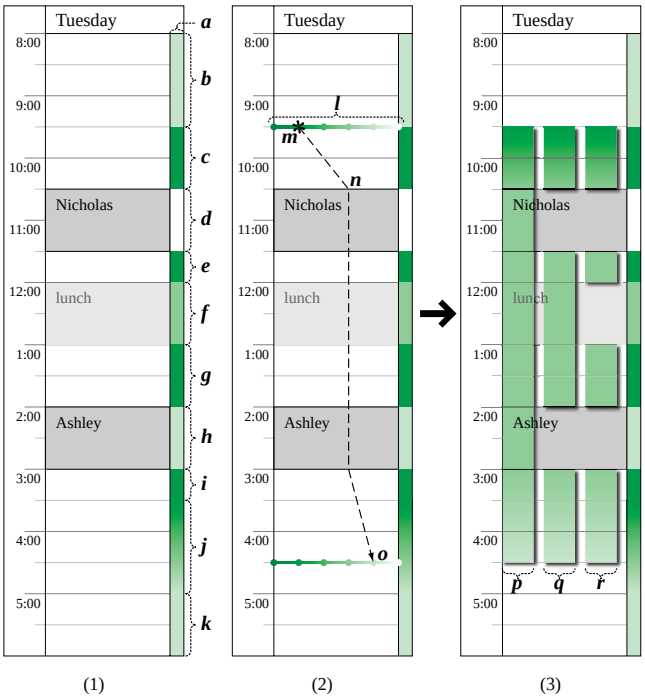

Figure 2. (1): The availability bar (*a*) shows a user's availability over the course of a day (*b*–*k*). (2): In a single gesture (*m*→*n*→*o*) the user can designate a region and apply a possibly time-varying availability preference level to it. (3): The original availability preference region (*p*) and two system-generated alternatives (*q* and *r*).

The technique works as follows: the user drags out a region of time in their calendar where the horizontal position of the mouse encodes the preference level at that time. A user enters the "painting availability" mode by selecting a paint availability tool from a toolbar or by issuing a command from a menu. A horizontal availability preference level bar (*l*) appears at the current vertical mouse position (*m*) with dots corresponding to each preference level, decreasing from left to right. The user initiates painting a region by clicking on the availability preference level bar with the mouse, which selects the preference level at the start of the region (*m*). Dragging the mouse allows the user to specify their preference level over time; the dashed line shows the path of the mouse. While dragging the mouse, the user can change preference levels by moving horizontally and clicking the mouse (*n*). The preference level is linearly interpolated between mouse clicks. To stop painting the re-

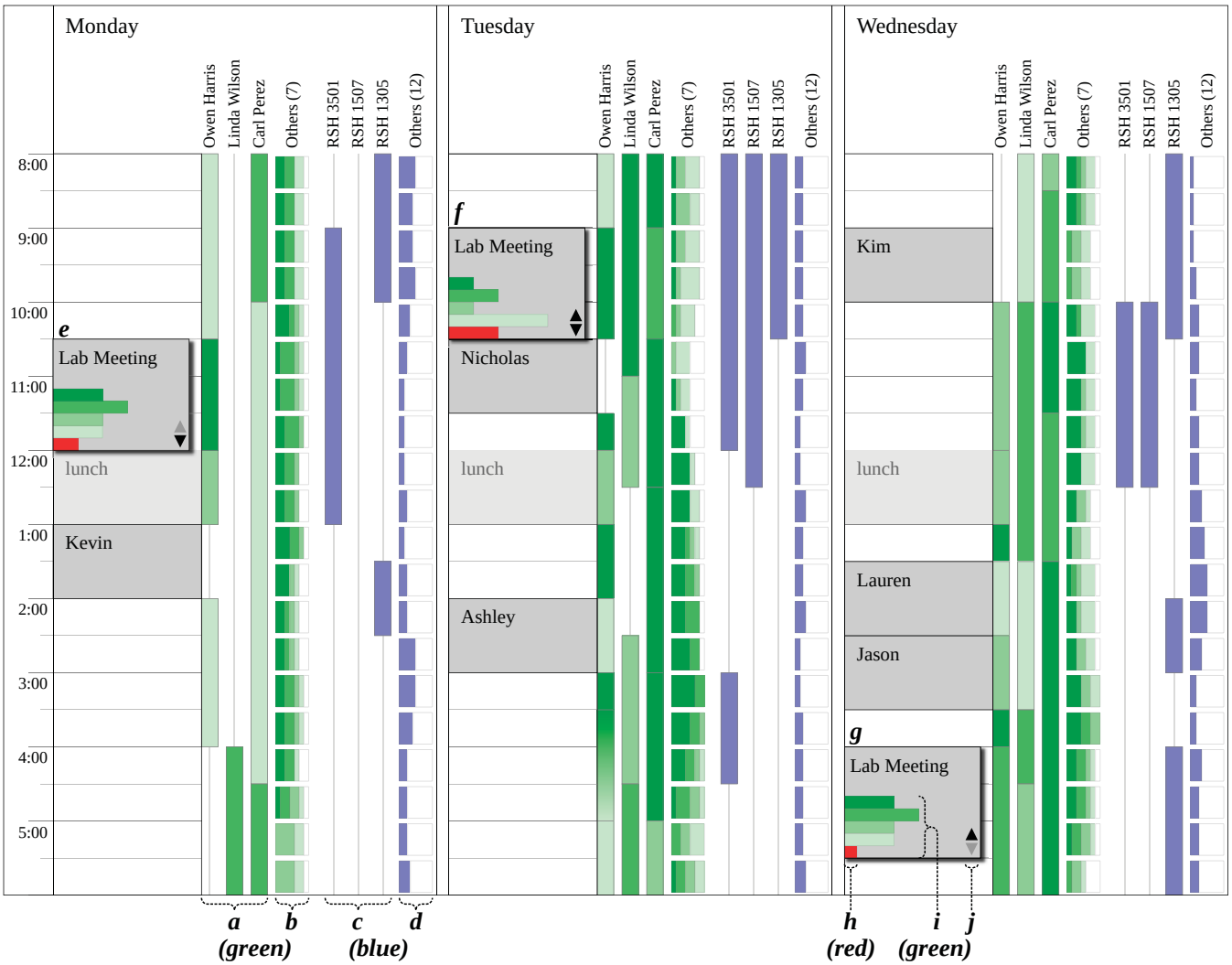

Figure 3. The display shows availability constraints of all invitees and the availability of conference rooms for a lab meeting. Three possible meeting times are shown (*e*, *f*, and *g*). At the bottom of each meeting, a histogram (*i*) shows the number of invitees per availability preference level. The red bar (*h*) draws attention to meeting options for which at least one invitee is unavailable.

gion, the user double clicks the mouse (*o*). The mouse stays in "availability paint" mode allowing the user to immediately paint another region. The user can edit these regions: changing their start and end times, editing how the availability preference level varies with time, deleting them, and copying them across days.

Once the user exits the "painting availability" mode, the display in Figure 2.3 appears with three availability options drawn as a set of translucent bars with shadows (*p*, *q*, and *r*). The left option (*p*) is exactly what the user dragged out, ignoring the availability bar for that day. The calendar application may also suggest alternative options (*q* and *r*). The middle option (*q*) is created by removing any scheduled meetings from (*p*). By offering this option, the user can quickly drag out a region, knowing that scheduled meetings can be easily excluded. The rightmost option (*r*) further excludes times that are unscheduled, but not available, such as lunch. The user can edit the options and if necessary, choose one to complete the response. If the user is responding to an email, the response would be converted to natural language and copied to the clipboard for pasting into a reply email.

5 Making Scheduling Decisions

We extended the availability bar technique to help a meeting organizer find an acceptable time and place for a group meeting with respect to all the constraints. An availability bar for each invitee is displayed side-by-side allowing a quick visual inspection to find promising times for the meeting to be scheduled. Even if the constraints have already been converted (when necessary) from natural language into a representation suitable for both optimization and graphical display, the constraints will still lack certain information needed for an optimizing algorithm to find the best meeting time and place. In particular, the constraint set may lack information about the relative importance of each constraint, such as how important a particular invitee's attendance is.

In the following scenario Owen, the meeting organizer, wants to schedule a lab meeting with nine other people sometime during the next three days. Figure 3 shows Owen's dis-

play after all the invitee's availability constraints have been collected. To the right of each day appears two sets of availability bars: the availability of each invitee (*a* and *b*: green hue) and the availability of conference rooms capable of holding ten people (*c* and *d*: blue hue). Note that people can have different availability preference levels, hence the different shades of green, whereas rooms are either available or unavailable: blue or white, respectively.[2]

Displaying availability bars for all ten people might require too much space, so the system displays availability bars for the three most important people (*a*) and a histogram summarizing the availability of the other seven people (*b*). The histogram shows the distribution of availability levels and offers more information than simply showing an availability bar with the average value.

The user can directly manipulate how many availability bars are visible. The user might even choose to see all availability bars, eliminating the summary histogram. The design also allows the user to see all the bars next to just one meeting. The user can change the order of availability bars, where the ordering corresponds to each bar's priority. The user could also group availability bars into priority classes. For example, it might be important that all invited faculty attend the meeting, but less important that their students attend. So the faculty would be in a group with higher priority than the student group.

Figure 3 shows three possible times for a "Lab Meeting" (*e*, *f*, and *g*).[3] Each meeting option appears in the foreground layer with a drop shadow to designate that it is proposed or tentative. A histogram at the bottom of each meeting (*i*) shows the number of people in each preference level. For the "unavailable" level, the histogram uses a red bar (*h*) instead of a white one to draw attention to the fact that some invitees cannot attend the meeting.

Highlighting one of the meetings will also highlight the other options for that meeting, which can be useful when multiple meetings are under consideration. The arrows at the bottom of each meeting (*j*) allow the user to move the focus to the chronologically previous or next option of the same meeting, which might be useful to show the user if there are options beyond the range of the current display.

6 Presenting Alternate Schedules

In the following scenario, a scheduling agent processes a meeting request and decides that it might make sense to reschedule an existing meeting. We describe visualizations that an agent might use to present alternate schedules.

In this scenario Laura asks Paula for a one-hour meeting at 2:00, though she mentions that she could meet any time from 12:00–4:30 if necessary. Paula already has three meetings scheduled (*a*, c_1, and *d* in Figure 4.1). At 9:00–11:30 Paula meets with Jerry (*a*), an important client, whose meeting cannot be moved or cancelled. At 2:30–3:30 Paula meets with Jeff

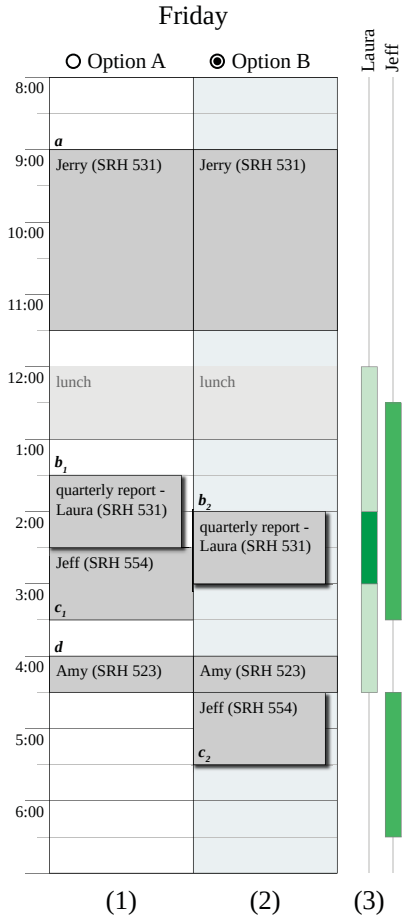

Figure 4. Paula's calendar. (1, 2): Visualizations of two alternative schedules for Paula side-by-side. (3): Laura and Jeff's preferences.

(c_1), who said that he would be available 12:30–3:30 and 4:30–6:30 when originally scheduling the meeting. At 4:00–4:30 Paula meets with Amy (*d*), her boss, whose busy schedule makes rescheduling this meeting difficult. As with prior examples, Laura and Jeff's availability constraints are shown on the right (see Figure 4.3).

Paula's calendar scheduling agent searches for schedules that satisfy the Laura's request and Paula's preferences. The system visualizes the agent's suggestions within the context of Paula's calendar to help her see how the proposal relates to existing meetings, and thereby relieves her of the difficulty of mapping natural language into calendar slots. The agent chooses to offer Paula two alternate schedules, which are shown side-by-side (see Figure 4.1–2). Meetings that the agent is proposing to add or change have shadows. The first schedule, "Option A," gives Laura the 1:30–2:30 slot (b_1). The second schedule, "Option B," gives Laura her preferred time (b_2) and bumps Jeff's meeting to 4:30–5:30 (c_2). In the figure, Paula has selected "Option B" using radio button at the top.

Since the system shows Jeff's availability preference, Paula can easily see that she could move his meeting later just in case the meeting with Amy runs long. Though, Jeff's availability preferences may have changed since his meeting was scheduled, so the agent might attempt to verify that they are still accurate. If the availability preferences were not verified, the system should display a warning to Paula (not shown).

[2] These visualizations could easily support variable availability of rooms, if the system had knowledge about which reservations might be movable.

[3] These times were chosen for their illustrative purposes; an agent would probably propose better times for this lab meeting.

7 Unknown Meeting Times and Durations

Current calendar tools do not provide support for visualizing and manipulating meetings with unknown start, end, or duration values. For example, a user may want to tell an agent to schedule a one-hour meeting on a given afternoon; the duration is known, but the start and end times are not. Similarly, the user may want to schedule a set of one-hour meetings for the afternoon, but does not care about their order.

A meeting with zero or more known values from among the start, end, or duration variables falls into one of five types (see Table 1). The three variables are related by a simple linear equation, *start* + *duration* = *end*, so it is not possible to have just two known values. Hence, a meeting can have 0, 1, or 3 known values. Type 0 (all values known) is the situation supported by current calendar tools. Type 1_{start} corresponds to meetings with a fixed start time and an unknown end time and duration. Type 1_{end}, symmetric to 1_{start}, corresponds to meetings with a fixed end time and an unknown start time and duration. Type $1_{duration}$ corresponds to meetings with a fixed duration but unknown start and end times, which is probably the most common situation with unknown values. Type 3 corresponds to a meeting with no known values.

While current calendars only support marking meetings as "tentative" or "pending," our visualizations support representing unknown information at a finer granularity. The design involves changing the left, top, or bottom border of a meeting to represent unknown duration, start, or end times, respectively. Figure 5 shows three designs for visualizing an unknown duration, (a)–(c), and four designs for visualizing an unknown start or end time, (i)–(iv). The twenty gray boxes in Figure 6 show how meeting objects appear when the visualizations for unknown duration and start/end values are combined. The top three rows show meetings of type 1_{start} using duration visualizations (a), (b), and (c), and the bottom row shows meetings of the type $1_{duration}$, in which the fixed duration is represented as usual by a left border with of a solid line. The columns show each of the four start/end visualizations.

Informally, we think that certain designs work better than others. We prefer design (b/i) due to its similarity and symmetry. Design (c/ii) has straight line segments that mesh well together. For start/end design (iii) both designs (a/iii) and (c/iii) have a nice visual appeal. However, duration design (a) has a very subtle appearance that may be visually lost in calendars with many appointments. Though, the visual subtly may be mitigated by the fact that unknown duration values will always be accompanied with unknown start or end values. We present the other options we have designed, in case others might find them useful. Formal user testing should determine which designs work best.

8 Related Work

Related work falls into three categories: collaborative human-agent optimization, calendar visualizations, and artificial intelligent technology.

The human-guided simple search (HuGSS) framework explores how a human can guide an optimization algorithm by suggesting paths that look the most promising [1, 18]. Our system focuses on collecting, visualizing, and prioritizing complex constraints, whereas with HuGGS the user is not involved in defining the constraints. The constraints are well-

Type	Known Variables	Notes and Examples
0	*start*, *end*, *duration*	Standard meeting with fixed times.
1_{start}	*start*	Meetings with a fixed start time that will run until a certain task is finished rather than until a specific time.
1_{end}	*end*	Meetings with a fixed end time, but an unknown start time. Allows the user to specify that a meeting must end by a certain time without specifying a start time.
$1_{duration}$	*duration*	For specifying a fixed length meeting without specifying exactly when. "I'd like a one-hour meeting with Tom."
3	none	For specifying that the user wants a meeting, but is not particular about the time.

Table 1. The five types in which zero or more of a meetings time variables (*start*, *end*, *duration*) can have unknown values.

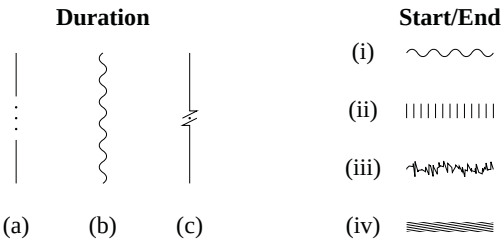

Figure 5. Various visualizations considered for representing unknown values for meeting times. (a–c): Three visualizations for an unknown meeting duration. (i–iv): Four visualizations for unknown meeting start or end times.

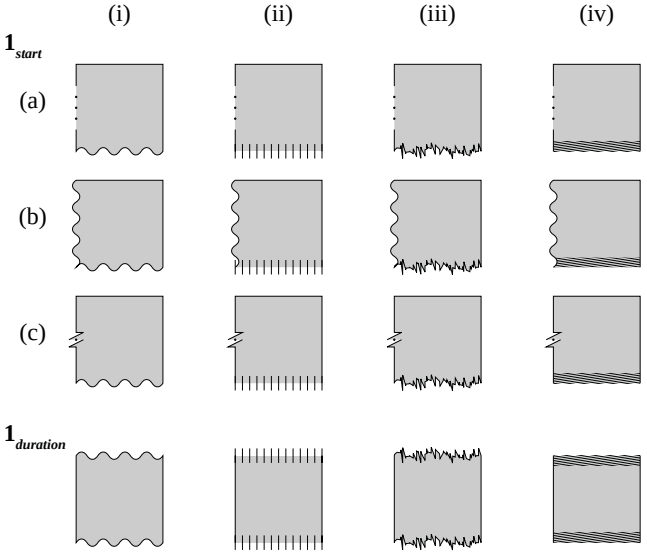

Figure 6. Various visualizations considered for representing meetings with unknown times. The top three rows show the appearance of meeting with known start time, but unknown duration and end times (type 1_{start}) using a combination of duration visualizations (rows) and start/end visualizations (columns) from Figure 5. Similarly, the bottom row shows the appearance of meetings with known duration, but unknown start and end times (type $1_{duration}$).

defined before the search process starts. Additionally, our visualization techniques should be useful even without an optimization algorithm.

Another system, VEIL, asks the user to choose among alternatives outcomes rather than explicitly describe their constraints [5]. The system's incremental utility elicitation (IUE) algorithm selects alternatives that should offer the most information. The IUE algorithm could be used to choose which alternatives to present to the user.

The Visual Scheduler [2] and Time Lattice [11] shared calendars help a meeting organizer identify common free times across multiple people's calendars. Both systems only support binary free-busy availability constraints, unlike our design's continuum of availability preference levels. Similar shortcomings occur in commercial shared calendar systems such as the Microsoft Exchange Server.

Several researchers have described calendar scheduling systems that focused on technical variations in artificial intelligence and neglected human factors in calendaring tasks [4, 7, 10, 12].

9 Discussion and Future Work

We plan to build a calendar system that incorporates these designs and deploy such as system within our environment. Following standard HCI methodology, we will include user testing and design iteration with the implementation.

9.1 Deployment

Building and deploying a calendar tool with these visualization techniques for our own use and for others within our organization should lead to many insights about what works in the real world since many members already perform complex calendar scheduling tasks. We expect to learn much about how these visualization techniques work in practice just as Bellotti *et al.* learned much from the development of their TaskMaster system for managing email [3]. Since users may be unwilling to abandon their current tools for research ones, any plan to deploy a calendar tool that uses our visualization techniques must include "fallback" to allow users to smoothly transition between our new calendar application and their existing calendar application.

9.2 Implementation Issues

In our experience, legacy applications provide reasonable programmatic interfaces to their data model, but do not allow significant modification to the user interface components required to implement our visualization techniques. The calendar applications commonly found within our deployment environment, Microsoft Outlook and Palm Desktop, do not provide adequate programming interfaces (APIs) for extending their calendar views for our needs. Implementing the visualization techniques requires the following functionality: (1) adding components to the layout, such as the availability bars and showing multiple schedule options per day; (2) extending the rendering of existing components, such as the visual layers and constraint representations for meeting objects; (3) adding event handlers to the event loop; and (4) adjusting component layout algorithms. However, Microsoft Outlook and Palm Desktop do provide a programming interface for reading and modifying the data model, in this case the collection of scheduled events.

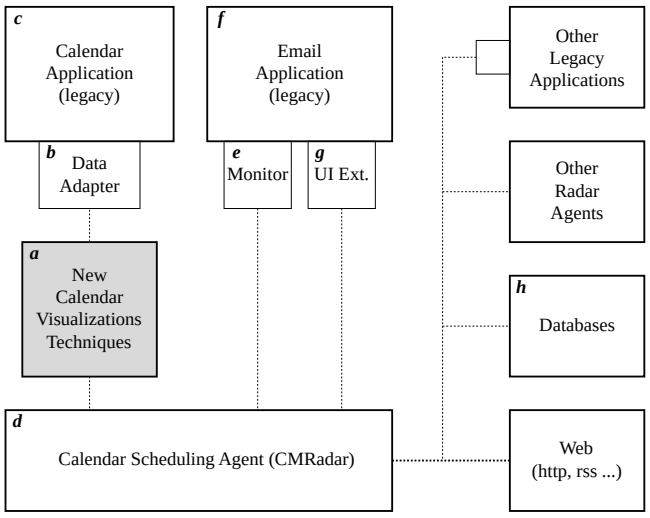

Figure 7. The "New Calendar Visualization Techniques" (a) uses a "Data Adapter" (b) to communicate with calendar applications (c) and an XML-based language to communication with CMRadar (d).

We will therefore implement our visualizations and interaction techniques within a calendar user interface developed from scratch that appears to the user to be another window of the application (Figure 7a). Data adapters (Figure 7b) will allow the new calendar user interface to read and modify the calendaring data model of the legacy calendaring application (Figure 7c). The adapters synchronize the data between the legacy and research systems, allowing the user to fall back to their legacy application when necessary. Users of different legacy calendar applications can use our new visualization techniques as long as a data adapter exists for their legacy calendar application. The communication language between new calendar user interface and the data adapter will be independent of the legacy calendar application, so we can more easily support multiple applications (for example, Microsoft Outlook and Palm Desktop calendars).

Our deployment will eventually be part of the full Radar system deployment. Figure 7 shows how our components will integrate within the Radar architecture. Our new calendar application (Figure 7a) will communicate with the Radar calendar scheduling agent, CMRadar [13] (Figure 7d) using an XML-based language that we are developing.

CMRadar or any other agent may need to monitor activity in other applications. For instance, CMRadar may monitor (Figure 7e) the user's email application (Figure 7f) looking for emails related to calendar scheduling activities. If CMRadar encounters such emails and needs to ask clarification questions, it can use our natural language clarification techniques [6], an example of a user interface extension (Figure 7g).

CMRadar would be able to access a database (Figure 7h) of room availabilities.

9.3 Other Domains

We also plan to apply these visualization techniques to other complex constraint-based search domains such as travel planning and product purchasing.

10 CONCLUSIONS

We identified several ways in which current calendar scheduling tools poorly support the task of scheduling complex meetings. We showed how this task is an instance of the general constraint-based search task. Better support for the calendar scheduling task requires techniques to author and visualize constraints. We showed how to embed the constraint authoring and visualization within direct manipulation style calendar application. This novel approach contrasts with traditional AI techniques that use chat-like natural language interfaces for eliciting constraints. Additionally, our visualization techniques should be useful even without an optimization algorithm.

We are presently working on designs to support scheduling recurring meetings. We will test all of our designs in lab studies to evaluate how well people can use them and to quantify performance improvements for common tasks. Then, we plan to build a calendar system incorporating our designs and deploy it within our organization to learn how the designs work in the real world, gathering information not available in lab usability studies. Finally, we also plan to apply our techniques to other domains such as arranging travel and making purchasing decisions.

11 ACKNOWLEDGMENTS

The authors thank Agata Adamowicz, Justin Cinicolo, Andrew Ko, Adam Lovrovich, Jay Modi, Jeffrey Nichols, Jean Oh, Yeming Shi, Stephen Smith, Desney Tan, Jacob Wobbrock and John Zimmerman. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the Department of Interior-National Business Center (DOI-NBC).

REFERERNCE

[1] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Brian Mirtich, David Ratajczak and Kathy Ryall, "Human-Guided Simple Search", In Proceedings of *Sixteen National Conference on Artificial Intelligence*, AAAI Press, Austin, Texas, USA, pp. 209–216, July 30–August 3, 2000.

[2] David Beard, Murugappan Palaniappan, Alan Humm, David Banks, Anil Nair and Yen-Ping Shan, "A Visual Calendar for Scheduling Group Meetings", In Proceedings of *Computer Supported Cooperative Work*, ACM Press, Los Angeles, California, USA, pp. 279–290, October 07–10, 1990.

[3] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, Ian Smith and Rebecca E. Grinter, "Quality Versus Quantity: E-Mail-Centric Task Management and Its Relation With Overload", *Human-Computer Interaction* 20(1/2), Lawrence Erlbaum Associates, pp. 89–138, June 2005.

[4] Pauline M. Berry, Melinda Gervasio, Tomás E. Uribe, Karen Myers and Ken Nitz, "A Personalized Calendar Assistant", In Proceedings of *AAAI Spring Symposium Series*, AAAI Press, Stanford, California, USA, pp. 7–12, March 22–24, 2004.

[5] Jim Blythe, "Visual Exploration and Incremental Utility Elicitation", In Proceedings of *Eighteenth National Conference on Artificial Intelligence*, AAAI Press, Edmonton, Alberta, Canada, pp. 526–532, July 28–August 1, 2002.

[6] Andrew Faulring and Brad A. Myers, "Enabling Rich Human-Agent Interaction for a Calendar Scheduling Agent", In Proceedings of *Conference on Human Factors in Computing Systems (Extended Abstracts)*, ACM Press, Portland, Oregon, USA, pp. 1367–1370, April 2–7, 2005.

[7] Melinda T. Gervasio, Michael D. Moffitt, Martha E. Pollack, Joseph M. Taylor and Tomas E. Uribe, "Active Preference Learning for Personalized Calendar Scheduling Assistance", In Proceedings of *International Conference on Intelligent User Interfaces*, ACM Press, San Diego, California, USA, pp. 90–97, January 9–12, 2005.

[8] Eric Horvitz, "Principles of Mixed-Initiative User Interfaces", In Proceedings of *Conference on Human Factors in Computing Systems*, ACM Press, Pittsburgh, PA, pp. 159–166, May 15–20, 1999.

[9] Charles L Isbell and Jeffrey S Pierce, "An IP Continuum for Adaptive Interface Design", In Proceedings of *11th International Conference on Human-Computer Interaction*, Las Vegas, Nevada, USA, 22–27 July 2005.

[10] Robyn Kozierok and Pattie Maes, "A Learning Interface Agent for Scheduling Meetings", In Proceedings of *1st International Conference on Intelligent User Interfaces*, ACM Press, Orlando, Florida, USA, pp. 81–88, January 4–7, 1993.

[11] Jock D. Mackinlay, George G. Robertson and Robert DeLine, "Developing Calendar Visualizers for the Information Visualizer", In Proceedings of *User Interface Software and Technology*, ACM Press, Marina del Rey, California, USA, pp. 109–118, November 2–4, 1994.

[12] Tom M. Mitchell, Rich Caruana, Dayne Freitag, John McDermott and David Zabowski, "Experience with a Learning Personal Assistant", *Communications of the ACM* 37(7), pp. 81–90, July 1994.

[13] Pragnesh Jay Modi, Manuela Veloso, Stephen Smith and Jean Oh, "CMRadar: A Personal Assistant Agent for Calendar Management", In Proceedings of *Sixth International Workshop on Agent Oriented Information Systems*, New York, New York, USA, pp. 134–148, July 2004.

[14] Pragnesh Jay Modi, Manuela Veloso, Stephen Smith and Jean Oh, "CMRadar: A Personal Assistant Agent for Calendar Management", In *Agent-Oriented Information Systems II*, Springer-Verlag, pp. 169–181, 2005.

[15] Jean Oh and Stephen F. Smith, "Learning User Preferences in Distributed Calendar Scheduling", In Proceedings of *Conference on Practice and Theory of Automated Timetabling*, Pittsburgh, Pennsylvania, USA, pp. 35–50, August 2004.

[16] Leysia Palen, "Social, Individual and Technological Issues for Groupware Calendar Systems", In Proceedings of *Conference on Human Factors in Computing Systems*, ACM Press, Pittsburgh, Pennsylvania, USA, pp. 17–24, May 15–20, 1999.

[17] Charles Rich, Candy Sidner, Neal Lesh, Andrew Garland, Shane Booth and Markus Chimani, "DiamondHelp: A Graphical User Interface Framework for Human-Computer Collaboration", Mitsubishi Electric Research Laboratories: TR2004-114, December 2004.

[18] Stacey D. Scott, Neal Lesh and Gunnar W. Klau, "Investigating Human-Computer Optimization", In Proceedings of *Conference on Human Factors in Computing Systems*, ACM Press, Minneapolis, Minnesota, USA, pp. 155–162, April 20–25, 2002.

[19] Anthony Tomasic, William Cohen, Susan Fussell, John Zimmerman, Marina Kobayashi, Einat Minkov, Nathan Halstead, Ravi Mosur and Jason Hum, "Learning to Navigate Web Forms", In Proceedings of *Workshop on Information Integration on the Web*, Toronto, Ontario, Canada, pp. 27–32, August 30, 2004.