Deploying a Personalized Time Management Agent

Pauline Berry, Ken Conley, Melinda Gervasio, Bart Peintner, Tomás Uribe, Neil Yorke-Smith
{berry,conley,gervasio,peintner,uribe,nysmith}@AI.SRI.COM
Artificial Intelligence Center, SRI International, Menlo Park, CA 94025 USA

ABSTRACT

We report on our ongoing practical experience in designing, implementing, and deploying PTIME, a personalized agent for time management and meeting scheduling in an open, multi-agent environment. In developing PTIME as part of a larger assistive agent called CALO, we have faced numerous challenges, including usability, multi-agent coordination, scalable constraint reasoning, robust execution, and unobtrusive learning. Our research advances basic solutions to the fundamental problems; however, integrating PTIME into a deployed system has raised other important issues for the successful adoption of new technology. As a personal assistant, PTIME must integrate easily into a user's real environment, support her normal workflow, respect her authority and privacy, provide natural user interfaces, and handle the issues that arise with deploying such a system in an open environment.

Categories and Subject Descriptors

H.4.1 [**INFORMATION SYSTEMS APPLICATIONS**]: Office Automation—*Time management*; I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence—*Intelligent agents*; K.6.3 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Software Management

General Terms

Design, Human Factors

Keywords

scheduling, personal assistants, user interfaces, deployment

1. INTRODUCTION

Over the past two-and-a-half years, we have been developing the Personalized Time Manager (PTIME), an agent with the goal of helping a human user manage her temporal commitments in a consistent, integrated manner over an extended period of time. While automated meeting scheduling has been well studied in the research community [8, 25, 23, 17], the resulting systems have rarely been adopted. We can point to two main reasons. The first is a failure to take into account the intensely personal nature of the human time management problem. Most individuals, even the busiest knowledge workers, are reluctant to relinquish control over the management of their own time. Further, people have different preferences and practices regarding how they schedule their time, how they negotiate appointments with others, and how much information they are willing to share when doing so. To be successful, a time management solution must thus address these basic issues of authority, privacy, and preferences. The second reason is that adoption of such a system is contingent on its practical usability—including important user interface issues—within a real-world deployment. While we do not claim to have fully resolved either one of these obstacles, this paper describes our experience and progress in overcoming these challenges.

Each PTIME agent is an autonomous entity that works with its user, other PTIME agents, and other users, to schedule meetings and commitments in its user's calendar. It can negotiate with other agents on behalf of its user, provide her with useful information about upcoming events while protecting her privacy, and learn her preferences. To achieve these goals, PTIME integrates commercially available calendaring tools and user interface technology with state-of-the-art algorithms for constraint satisfaction (to solve the underlying scheduling problems and find preferred schedules), goal-directed process management, and preference learning.

PTIME is part of the CALO project, a large-scale effort to build an adaptive cognitive assistant situated in the office environment [15]. A CALO agent is designed to support its user in various ways, including project and task management, information collection and organization, and meeting preparation and summarization. In October 2005, the CALO project started its third year of research and development. In Year 1, the theme was *integration*—assembling a large collection of independently developed pieces of software into a single system. CALO 1.0 was highly distributed, with the CALO modules implemented on top of the Open Agent Architecture (OAA) [4] and with integration primarily at the level of communication. CALO 1.0 required multiple computers to run and provided only the most rudimentary desktop functions. In Year 2, the theme was *usability*—integrating the applications within a common architecture that could run on a single laptop computer. CALO 2.0 was reborn within the IRIS desktop environment [5], which featured a semantic knowledge base, based on the CALO ontology, and shared by a suite of common desktop applications, including email, calendar, web, file system, and chat. Through the shared knowledge base, IRIS supported the sharing of information between the disparate applications, enabling CALO to better control and coordinate its assistive behavior. With CALO 2.0, users could read email, file documents, schedule meetings, organize bookmarks,

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

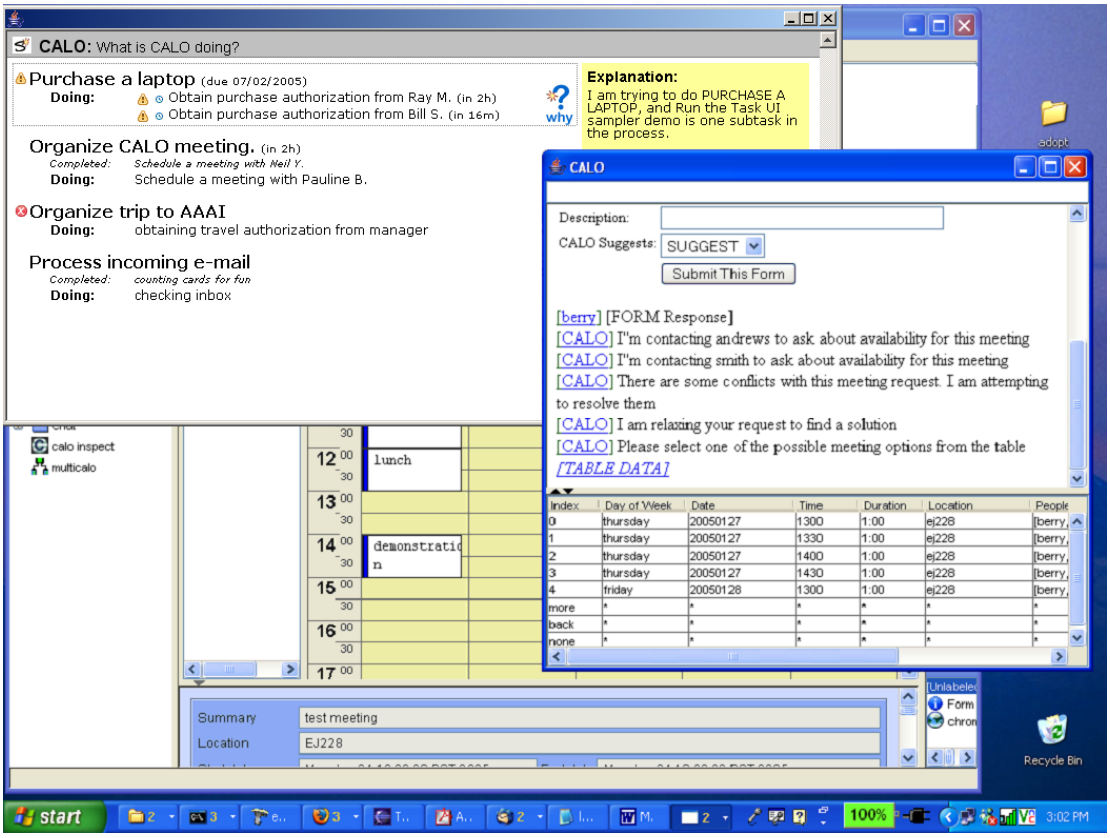

Figure 1: Screenshot of Year 2 PTIME interface

and perform various other everyday office activities. However, there remained obstacles to its widespread use, such as cumbersome user interfaces and computational inefficiency. The theme for Year 3 is *usefulness*—providing users with a CALO that is not only usable but can also truly help make busy office workers more efficient and effective in their everyday lives.

This paper describes lessons learned from the development and deployment of PTIME within the CALO project. We begin by presenting the PTIME system, its local agent architecture, and its deployment in a multi-agent setting (Section 2). Next, we present the challenges we faced and the solutions we implemented to successfully field a PTIME agent in CALO 1.0 and 2.0 (Section 3). Then we discuss other issues that arose post-deployment, through actual use by real users, and we present the modifications we made to address these issues prior to the final, annual release (Section 4). We then discuss our plans for CALO 3.0, including the modifications to existing features and implementation of new features designed to address known issues as well as new challenges (Section 5). We conclude with a discussion of related work and a summary of lessons learned (Sections 6 and 7).

2. PERSONALIZED TIME MANAGEMENT

Despite the proliferation of calendar tools to organize, display, and track commitments, most individuals still expend a considerable amount of effort managing their schedules.

The research community has primarily focused on automated meeting scheduling: finding feasible timeslots for meetings given a set of requirements on participants, times, and locations. Work in this area can be generally divided into *open* and *closed* scheduling systems [8]. In an open system, individuals are autonomous, responsible for creating and maintaining their own calendars and meeting schedules. They generally operate in an unbounded environment, without any obligation to a single organization. In a closed system, by contrast, a consistent and complete global calendar is maintained. While closed systems are more commonly studied and marketed because they provide a controlled and known setting, they are rarely adopted in a uniform way because users seldom live in a completely closed environment. However, open scheduling systems pose challenges of their own. An individual may not wish to share all of her schedule, or may choose not to participate in a meeting. Different individuals will have different scheduling preferences. Without centralized control to balance the constraints and preferences of all involved parties, how does an open system respect individual preferences without devolving into completely selfish agents that result in greatly suboptimal schedules? Further, while users may not wish to cede all control to an autonomous assistant, they may let their agent automate some processes.

CALO exists in such an open, unbounded environment, where these issues of privacy, authority, adjustable autonomy, cross-organizational scheduling, and uncertain availability of participants abound. PTIME is an autonomous agent integrated within a larger CALO agent [15]. Each CALO/PTIME agent is itself composed of a set of internal agent components linked within the OAA agent architecture; we will refer to these as CALO/PTIME components.

2.1 The Distributed Scheduling Task

The distributed scheduling task that PTIME addresses fits into the continuum between fully competitive and fully cooperative agents. PTIME agents in collaboration with their users are autono-

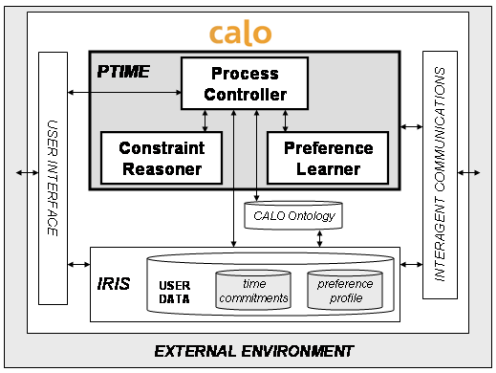

Figure 2: Architecture of Year 2 PTIME agent

mous, self-interested entities that manage their user's own calendars. They are in effect single-calendar schedulers dependent on other agents to coordinate shared calendar entities. The scheduling task is viewed as a shared goal of the user and the agent. For a PTIME agent, global utility is thus secondary to maximizing the local utility of its user, which typically does include a component valuing cooperation with others [22]. This is in contrast to most distributed scheduling systems [25], which aim to maximize some notion of global schedule quality (e.g., [16]).

Thus, PTIME supports two forms of collaboration: between itself and its user and between itself and the PTIME agents of other users. The collaborative scheduling process is separated from the scheduling algorithms, to enable interaction with the user and other PTIME agents. This interaction forms the framework for learning and adjustable autonomy.

The collaboration between PTIME and its user is illustrated in Figure 1, which is a screenshot of the PTIME interface for the deployed CALO 2.0 system. The collaboration is seen in the "CALO Chat" window in the foreground (right). In the top-left window, CALO is reporting on its activities in response to a user request for explanation. In the background, the calendar view of the IRIS desktop environment can be partially seen. In addition to a traditional form-based input mechanism, CALO has a restricted natural language (NL) input that allows the user to interact by means of spoken or typed input, for instance "meet with Alice about integration next week". CALO also has multi-modal output, including leaving spoken messages on the user's cellphone.

2.2 Local Agent Architecture

Figure 2 depicts the architecture of PTIME in relation to the relevant components in IRIS and CALO. We limit our discussion to the three PTIME components this paper primarily involves; for a more complete discussion of the various components of PTIME, see [3].

- At the heart of PTIME is its *Process Controller*, a SPARK agent that captures possible interactions between the user and other agents in the form of structured decision points. SPARK [19] is a Belief-Desire-Intention agent framework that supports both goal-directed and reactive behavior. The Process Controller manages PTIME's processes, tasking and coordinating the activities of the Constraint Reasoner and Preference Learner, and managing interactions with the rest of CALO and the external environment. Alternative processes reflect different scheduling strategies, preferred by users or organizations, or applicable in different contexts. For example, different processes would correspond to scheduling a set of interviews with available interviewers versus allowing interviewers to sign themselves up. The SPARK framework supports agent advisability [20], and users can give advice to PTIME regarding preferred actions as well as explicit scheduling preferences. SPARK also supports adjustable autonomy, and users can explicitly or gradually delegate certain decisions to PTIME for autonomous execution.

- The basic scheduling process employs a unified representation for temporal and nontemporal constraints, and preferences. Unlike standard calendaring systems, as part of CALO, PTIME has access to rich information about the user's other activities. Thus, even when scheduling simple meetings, the Constraint Reasoner considers such issues such as workload and task deadlines, to ensure, for instance, that the user is not over-committed. The Constraint Reasoner can also explore alternative conflict resolution options using relaxation, event bumping, and explanation techniques [14].

- A core design goal of CALO is personalization. PTIME evolves by learning preferences through its *Preference Learner*. The Preference Learner in PTIME is an unobtrusive, online learner where the user's selections from suggested alternatives provide feedback to the learning algorithm. Learning has so far focused on temporal scheduling preferences over when meetings should be scheduled (e.g., time of day, day of week) and over the nature of the resulting calendar (e.g., degree of fragmentation). PTIME learns these preferences for under-constrained problems. We have showed how active learning with support vector machines (SVM) can be used to improve performance [11]. Learning a user's preferences enables PTIME to suggest increasingly desirable schedules to the user, thus providing more effective assistance.

The components of a PTIME agent utilize a variety of programming languages: the Process Controller is implemented in SPARK, the Constraint Reasoner in SICStus Prolog [27], and the Preference Learner in Java, with its underlying SVM learner being SVMLight [13], which is implemented in C. The different components, deployed as OAA agents, communicate with each other through the mechanisms provided by the OAA architecture [4].

3. DEPLOYMENT CHALLENGES

Deployment of the PTIME calendaring system required overcoming several multi-agent integration issues. We discuss them in this section, together with some of the deployment constraints imposed by the CALO personal assistant system.

3.1 Calendar Integration

Most fundamentally, a calendar scheduling system must be able to update and maintain a user's calendar. PTIME uses the *iCalendar* specification [7] for compatibility with other calendar tools on the market. An initial implementation used Microsoft Outlook, which was instrumented so that PTIME could query information from the calendar; add, delete, or modify calendar events; and listen to calendar actions performed by the user. Initial experiments with test users indicated this was a workable integration. One problem with this implementation, however, was that availability information was inaccessible when one or more users did not have their desktop environments connected.

A second integration was required with the introduction of the CALO user environment, IRIS [5], which includes a calendar application for accessing a centralized commercial calendar server.

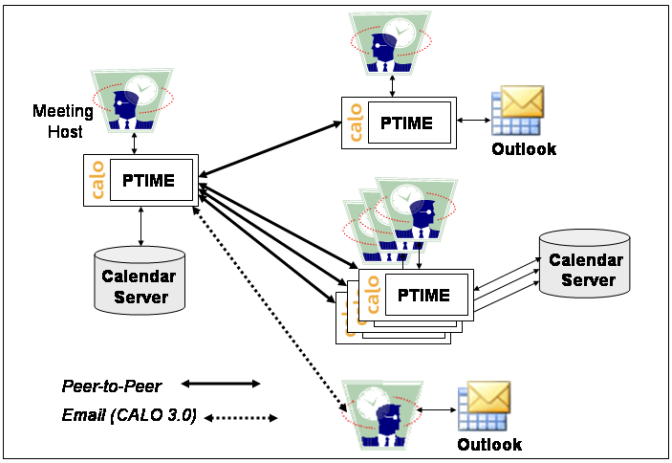

Figure 3: Communication modes between PTIME agents

The IRIS calendar application allowed PTIME to query user calendar information using the Glow API [1], but many users preferred to continue using their familiar calendaring tools, such as Outlook or Mozilla, that do not store their data on the centralized calendar server. There were also data synchronization issues, since PTIME could query the server for a participant's calendar status even when that user's CALO was unconnected.

One particularly challenging architectural constraint was that we could not make modifications to the calendaring user interface in IRIS. A core CALO 2.0 design decision was to keep "CALO actions" completely separate from the regular operations of the IRIS applications. Instrumentation of existing dialogs and development of external user interfaces were allowed. The result of this architectural constraint was two separate interfaces for adding meetings to the calendar: the standard "Add meeting here" Outlook-style dialog within the IRIS calendar application, and a separate CALO interface for scheduling through PTIME.

3.2 Multi-Agent Integration

A PTIME agent must communicate with other people's calendars to gain availability information, and negotiate with other agents to resolve conflicts and commit to calendar changes. Calendars are private; we assume that direct interrogation of other users' calendars is restricted to free/busy timeslots. Other calendars cannot be updated directly without adequate permission. PTIME agents communicate using peer-to-peer messaging or using email (Figure 3). The initial deployments (CALO 1.0 and 2.0) relied only on peer-to-peer communication via message passing. The next deployment will also offer email communication peer-to-peer as well as with the user. Thus, PTIME will be able to schedule with any online participant, whether or not she is running CALO. The protocol adopted for inter-agent negotiation is the following:

- Use direct agent-to-agent communication: resulting action would be governed by the destination's own agent.
- Agent not answering: for availability requests, ask the calendar server for a recent (possibly outdated) free/busy estimate.
- Otherwise: send an email query that could be answered by the human user or the other (PTIME) agent.

This communication protocol exists in a deployed and unbounded environment where not all participants in a meeting may be using PTIME, the shared calendar server, or indeed any automated calendar. In addition, we cannot assume that the participant's contact details exist in the user's address book. During the initial deployment of the system, a simple name resolution system asked the user for clarification of a participant's email address. However, we encountered a problem when a person had multiple email addresses and thus PTIME would duplicate messages. Although we did work around this problem, for current and future deployment we have a two-pronged solution:

- When a CALO agent (of which PTIME is a component) is initialized it sends information to a common name server. This name server stores the contact information for other agents, and this information may be queried by any agent.
- The CALO agent also provides a name resolution capability that allows a participant's name to be resolved to a known person on the name server, in the personal CALO address book, or an email address typed in by the user.

3.3 Knowledge Integration and Advisability

In addition to personal and shared 'contact' information, a personalized time management assistant must also have access to a user's organizational information, institutional rules, and personal preferences. For example, a PTIME agent may schedule a meeting with the user's superior or subordinate using different strategies. PTIME can store locally some of this information, such as scheduling preferences, but it must access other information, such as organizational relations, from the wider CALO knowledge base. PTIME adopts the CALO-wide ontology and query standards.

A foundational requirement for deployment was user–PTIME interaction. Not only must PTIME serve the user's needs in a mixed-initiative fashion [3], but it must also become a more effective assistant over time. It must build trust through its interaction with the user. Thus, PTIME permits the user to provide advice [20] that adjusts the level of autonomy, and permits her to override decisions made by the agent. A specific problem with these simple mechanisms to adjust autonomy was that the users were reluctant to change settings if the system did not provide an adequate explanation of their effects. For example, the user might advise the agent about preferred days of the weeks for meetings, but PTIME did not have the ability to illustrate to the user the resulting preference profile or to explain decisions with respect to those preferences.

4. LESSONS FROM TESTING AND USE

The CALO program is evaluated annually through a SAT-style test administered on a number of CALOs that assist users over some designated period of time. In Year 2, the test involved fifteen users, including some of the authors of this paper, using their CALOs over the period of a week to perform a variety of everyday office tasks, including tending to email, scheduling and preparing for meetings, editing documents, and performing web searches.

In preparation for the test, CALO underwent a standard software development cycle involving a series of releases each featuring increased and more robust functionality. As part of this process, CALO underwent quality assurance testing, including regression and unit tests. Each release was vetted through "dry runs" with real users. The process provided valuable feedback to the research and development teams, validating some design decisions while raising new problems, allowing time to address them before the actual test.

4.1 Different Modes of Use

PTIME was initially designed to schedule meetings on behalf of the user, including accepting or declining meetings scheduled by

other users. However, users turned out to desire PTIME support for at least two other usage modes as well.

First, users often simply wanted to add fully-specified events (e.g., seminars, weekly group meetings) to their calendars regardless of overlaps with existing events. PTIME's standard scheduling interface could be used to add such meetings, resulting in a single alternative being presented for acceptance by the user, with the option to reschedule in the case of conflicts. However, with this solution, what should have been a one-step procedure turned into a multi-step process, because of the verifications required from the user. The offer to reschedule in case of a conflict was also inappropriate for these types of events, which were already scheduled.

The second situation involved scheduling a complete agenda or related set of meetings centered on another person, for example, scheduling an interview for a job candidate. Here, the user's preferences are of no special import, since the schedule is for someone else. There may also be constraints and preferences not just on individual meetings but between meetings: for example, minimizing the free time between meetings so that the visitor is not left idle. We handled this scenario by implementing a separate set of SPARK processes for visitor scheduling, including separate facilities for obtaining the scheduling constraints from the user, and presenting agenda options. This worked reasonably well, and we will continue to do this for CALO 3.0.

This second usage mode also revealed inefficiencies in the Constraint Reasoner. While efficient enough to find alternatives for scheduling one or a few new meetings at a time, the Constraint Reasoner was often unable to efficiently find a solution in the significantly larger search space generated by visitor scheduling scenarios, consisting mostly of temporal constraints. Our solution to the search inefficiency problem was less than desirable: we modified the constraints formulated by the agenda scheduling problem to reduce the size of the search space sufficiently so that the Constraint Reasoner could find a solution if one existed. Clearly, this is only a stopgap solution since it forces users into unnatural and inexact problem formulations. We have solved this problem for CALO 3.0 by using constraint-solving techniques specifically designed for temporal constraints (discussed in Section 5.3).

4.2 Visibility into the Scheduling Process

Scheduling a meeting involving multiple participants can be a somewhat involved process, as it requires gathering availability information for all participants, finding valid solutions or—possibly—relaxing unsatisfiable constraints, before presenting the user with candidate solutions. Early on, it became clear that we needed to give users some method of tracking the scheduling process. Thus, as it undertakes the various tasks involved in scheduling a meeting, PTIME prints simple status messages in the CALO chat window to inform the user of its progress, as seen in Figure 1.

This feedback proved valuable in letting the user track PTIME's progress, particularly when unexpected situations arose. For example, obtaining another person's availability information typically takes a few seconds. In the case of delayed results, whether due to the other person wanting to personally approve all requests or due to inter-CALO communication failure, PTIME could notify the user accordingly, letting her take action if desired.

A related issue is CALO's ability to explain its scheduling decisions. Often, users want to know why CALO is unable to schedule a meeting. We addressed this by providing a minimal set of unsatisfiable constraints [14] when informing the user that a particular meeting request was unsatisfiable. This explanation mechanism also turned out to be particularly useful when debugging the system, exposing, for example, the Constraint Reasoner's flawed expectation that all events have non-zero durations. Similarly, when CALO suggests relaxed alternatives, users often want to know what constraints have been left unsatisfied so that they can make well-informed decisions. We will be extending the explanation mechanisms in CALO 3.0 to cover this situation as well, and linking PTIME's explanations with the broader CALO-wide explanation capability seen in the left-most window in Figure 1.

4.3 Qualitatively Different Schedules

Since our learned schedule evaluation was based on simple weighted sums, the best schedules tended to be clustered together. One of the factors we investigated in our active learning experiments was the presentation of qualitatively different schedules to the user, motivated by the intuition that users would be interested in seeing a variety of schedules and that the resulting feedback would also improve learning. However, as our experiments showed, the greedy approach of showing the best solutions according to the current learned model resulted in the fastest learning [11]. As a result, the fielded system used this approach to select candidates to present.

While most successful for learning, this scheme often resulted in highly similar schedules being presented to the user. For example, it was not uncommon for PTIME to present alternative candidates that began at 9:00, 9:30, 10:00, 10:30, and so on. This was particularly problematic early on, when PTIME had not yet sufficiently learned the user's preferences and was thus unable to present any good schedules within its default presentation set of 15 candidates. In our experiments with synthetic users we had relied on the user's ability to override bad solutions by scheduling the meeting manually. However, user interface limitations (see Section 3.1) prevented such a solution: users could schedule the meeting manually through the regular IRIS calendar interface, but in bypassing PTIME, the learning algorithm could receive no feedback on its suggestions. We believe a second issue is that our synthetic users did not fully represent the different kinds of preferences real users have; in particular, preferences between criteria. We are currently developing a richer, multi-criteria evaluation approach (Section 5.4) that we expect will lead to better adaptation.

4.4 User Interface Issues

Our primary focus for CALO 2.0 was scheduling functionality: encoding the procedures controlling user interaction and the scheduling process, implementing the constraint solver to generate solutions, and developing the learning module to provide personalized scheduling assistance. As a result, PTIME's user interface was functional but not particularly effective. Besides being hard to use, it prevented us from exposing all of PTIME's functionality.

First, PTIME provided a single dialog form through which users could specify a meeting request. This was a particularly poor fit to other scheduling situations. Moreover, it was also a poor fit to basic meeting scheduling, as it required users to specify constraints differently to their normal way of specifying meeting requests.

Second, while users could explicitly state general scheduling preferences to bootstrap the learning process, the mechanism was both cumbersome and unable to handle fine-grained preferences.

Third, PTIME presented its candidate schedules in a table (as seen in Figure 1), completely separate from the IRIS calendar interface. This was primarily due to the architectural constraint of keeping CALO actions distinct from the regular operations of the desktop applications. However, it turned out to be problematic in this case as it not only confused users, but also presented the candidate schedules almost completely devoid of context. Without knowing their existing schedules, users often found it difficult to select an appropriate candidate schedule.

5. ONGOING/FUTURE DEPLOYMENT

The third annual release of the CALO agent system, CALO 3.0, will deploy a new version of PTIME that will address the lessons learned from CALO 2.0 and add new capabilities to the agent. The ultimate goal is to advance the capabilities and interface to a point where people other than PTIME developers and CALO testers find value in using their CALO agent on a daily basis. Below we outline five improvements designed to bring us closer to this goal. CALO 3.0 will be evaluated similarly to CALO 2.0; the increased usability will facilitate the next round of user studies, which, along with daily use, will surely uncover new lessons about developing and deploying user-assistive agents in unbounded environments.

5.1 Multiple Modes of Use

Recall that the PTIME user interface in CALO 2.0 enabled the interactions necessary for PTIME's advanced functions, at the expense of encumbering the user with extra steps for the simple, more common, scheduling scenarios. We are addressing this issue in CALO 3.0 by defining distinct use cases and implementing a different SPARK process for each case, including different input and output facilities in the user interface.

The new version of PTIME will allow users to simply add meetings to their calendars, with or without checking for overlap and communicating meeting information to participants. In the case of overlap checking, PTIME's course of action is still an open issue. Ideally, if the user is looking at the appropriate portion of her calendar, then an overlap should be obvious and the user would not want to be bothered by superfluous confirmation requests or warnings. Instead, a simple highlighting may occur. In the more complex case of multiple participants, we must determine whether PTIME should check for participant conflicts and whether PTIME should pass on the commitment to the participants. PTIME may do background checks for such changes in the user's calendar and warn the user about conflicts with or rejection by other participants.

5.2 Multi-User Scheduling

Our informal user studies confirm that other participants' constraints and preferences often influence the meeting organizer's decisions [22]. When deciding which of many possible candidate schedules to present to the user, PTIME must reason about how each candidate schedule will affect others. This requires PTIME to retrieve from each affected party that person's schedule (as in CALO 2.0), preferences, and probable effects of the proposed schedule. Moreover, reasoning about other participants is required even when communication with another user's agent (CALO) is not available—and hence, should a shared calendar server be in use, even when the information it contains is unreliable (recall Section 3.1). Once a PTIME user selects a schedule from those presented, PTIME must negotiate with the agents of the other parties to resolve conflicts within each schedule. The goal is to remove the burden of much of this reasoning and negotiation from the user, while keeping her informed about the state of the process.

To determine the effect of a particular schedule on others, we are currently integrating a distributed querying mechanism, based on the ADOPT protocol [16], that will obtain a cost reflecting the rescheduling effort required as a result of a proposed schedule. The cost for users who do not respond (i.e., communication with their CALO fails) is estimated using the most recent knowledge about their schedules and their negotiation history.

When the user organizing a new meeting selects a candidate schedule, PTIME invites all other meeting participants, with a justification for the request. All conflicts are then addressed by other users, who will either resolve the conflicts with other CALOs or choose to not attend. Negotiation with the meeting organizer to counter-propose an alternate time is planned for CALO 4.0.

5.3 Improved Constraint Reasoning

Section 4.1 described some of the limitations of the constraint reasoning in the CALO 2.0 system. The current technology will be replaced with a Constraint Reasoner efficient for both finite-domain and temporal constraints [26, 18]. Increased performance improves the user's interaction experience with CALO, and allows PTIME to present more varied candidate schedules (Section 4.3). For instance, in the scenario of scheduling a complete agenda (e.g., for an interview visit) described earlier, preliminary experiments show that computation time can be reduced by orders of magnitude, putting larger multi-event scheduling problems within reach.

Further, the new technology will allow representation of soft constraints, enabling the user to specify, for example, preferred times for meetings, the relative importance of each participant, and events that may partially or fully overlap. In the case where many schedules satisfy all hard constraints, the soft constraints allow computation of schedules that best satisfy the user's preferences. In the case where no schedule satisfies all hard constraints, the soft constraints provide instruction for how to best relax constraints. For example, knowing the relative importance of each meeting participant would guide the Constraint Reasoner to first try omitting an unimportant participant before relaxing other constraints.

5.4 Schedule Evaluation Criteria

For meeting requests other than the simple "Add meeting here" type, PTIME presents multiple candidate schedules to its user. As stated earlier, in CALO 2.0 the candidates were ranked according to the learned schedule evaluation function, a simple weighted sum over schedule features, and a subset was selected for presentation by an active learning mechanism. Based on the results from actual use, informal user studies, and interviews of subjects from the CALO 2.0 test, we determined that this approach was inadequate for a number of reasons. The most important is its inability to express tradeoffs users have in practice: for instance, Pareto optimal schedules that balance two criteria [2].

Based on those same studies, we have identified key criteria that contribute to the user's evaluation of a schedule. These features include satisfaction of the user's general preferences on schedules, specific meeting-request constraints (both hard and soft constraints), perturbation from the previous schedule, and, as noted, the goodness of the schedule according to others. We will use these criteria in a multi-criteria evaluation framework [2] in which the relative importance of each criterion is learned.

When deciding which solution candidates to present, PTIME has multiple objectives: present desirable schedules that the user is likely to accept; present qualitatively different schedules to help the user explore the space of possibilities; and support faster learning of preferences over criteria. The second point is particularly important if the user has not yet formulated what is a 'good' solution to a specific meeting request. Only after the user has elicited in her own mind what is 'good' can we address the other two points, presenting desirable schedules and learning preferences. We are now exploring search techniques that can produce not only a single optimal solution, but also a good set of qualitatively different solutions to meet these objectives. Based on an example critiquing approach, in which solutions are presented to the user to prompt her decision process, we will present schedules designed to maximally stimulate expression of the user's preferences [28]. In conjunction, we are exploring how to vary the diversity of the solutions chosen for presentation based on a measure of entropy of the solution set.

Figure 4: Concept of future PTIME interface

5.5 Collaborative Interface

User feedback from CALO 2.0 reinforces the point that an intuitive and visually appealing user interface (UI) not only enhances human–agent collaboration, but strongly influences whether users are willing to adopt the underlying agent technology. To encourage adoption of PTIME in CALO 3.0, we will redesign the UI to improve the two main tasks: defining a meeting request and viewing/selecting candidate schedules for that request. This new UI is being designed and developed in collaboration with human-computer interaction professionals.

When defining a meeting request, PTIME's interface will allow more expressive, albeit restricted, natural language input. Although specifying preferences such as "prefer Tuesday at 2pm", "Bob is optional", and "can overlap" is possible with dialog form widgets, to include the appropriate widgets for all possible constraints and preferences would result in an overly complex interface, and such interfaces have been found to elicit untrue preferences [28].[1]

When candidate schedules are presented to the user, PTIME will display a collaborative, graphical interface to help the user rapidly understand the key differences among the set of schedules, refine or relax her input constraints, explore alternative solutions, and select a preferred candidate. This will replace the generic text-based table used in CALO 2.0.

Figure 4 shows a mockup of the future interface, including the visualization for presenting candidate schedules. "Overlaps with Coffee" (lower left-corner) indicates to the user that some hard constraints were not fully satisfied; red underlines indicate portions of the user's schedule that had to be rearranged. In a separate editing mode, PTIME will show what rearrangements are compatible with other users' schedules. The top half of the interface displays past actions of both the user and CALO to provide additional context.

[1]Users often fill out all the widgets on the form, even those marked as optional: simply asking whether the user has a preference for a feature can lead to the user having a preference for it.

6. RELATED WORK

While both commercial (e.g., Outlook/Outlook Exchange Server, Sun Java System Calendar Server) and open source (e.g., Zimbra, UWCalendar) calendaring systems abound to support centralized solutions within an institution, as closed systems [8] they suffer from the issues discussed earlier, including resistance to adoption due to loss of control and privacy. Unlike PTIME, they offer a one-size-fits-all solution that does not readily support different calendars, different modes of interaction, and different user preferences—often exhibited in the necessity of an institution-wide mandate to standardize on a particular tool. Also, unlike PTIME, which is part of a much larger cognitive assistant (CALO), these tools are dedicated calendar scheduling solutions that cannot utilize other types of information—such as workloads and deadlines—to better manage the user's schedule. Similarly, tools such as OfficeTracker and GroupReady focus on visualization and facilitating a scheduling process actually performed by a human.

In the applied research community, scheduling agents have been developed for fragments of the tasks with which a PTIME agent assists a user (e.g., [12]). Examples of other scheduling systems that operate in open environments include RCal [23] (arguably the first agent work for this scheduling task), CMRadar [17], and work in preference-aware constraint programming, such as [10].

The CMU scheduling agents RCal [23] and CMRadar [17] both distribute the scheduling problem in a similar way to that of PTIME, and the more recent CMRadar embodies a spirit similar to PTIME. A CMRadar agent helps its user schedule meetings by parsing out meeting information from email. It generates schedule options using a constraint-based scheduler that takes into account user preferences; presents candidate schedules with a graphical visualization; and enters into multi-agent negotiation. CMRadar is designed as a standalone scheduler and is tightly integrated into Microsoft Outlook, while PTIME is integrated into CALO and primarily uses the IRIS interface to access external calendar servers. PTIME is also able to support other individual calendar tools, and schedule meetings with non-CALO-related individuals. In a further architectural difference, CMRadar relies solely on email for inter-agent communication, while PTIME uses multiple modalities, with direct inter-CALO communication as its primary modality. CMRadar uses a passive learning approach to learn user preferences [21], which contrasts with PTIME's online, active learning.

Another mixed-initiative, multi-agent scheduling system is currently deployed in a closed environment to schedule nightly maintenance operations for the Hong Kong metro system [6]. The system provides core scheduling abilities and visualizations that facilitate both the scheduling of operations and the negotiation of resources between managers. Like PTIME, the UI for this system allows a window into the workings of the automated scheduler.

The CALO 3.0 UI offers multi-schedule comparison similar to RhaiCAL [9] and several non–agent-based commercial calendaring tools (e.g., GroupReady), although PTIME's visualization is part of a larger collaborative user–agent dialogue paradigm. Nonetheless, the independent development of such UIs underlines key interaction challenges, which include how an agent can ask a user to check and correct its understanding of NL input, and how it can iterate with a user to approve, reject, or modify proposed actions. Explanation in the context of collaborative and semi-autonomous assistive agents has been explored in frameworks such as Collagen and the (prototype) deployed systems built with it [24].

7. CONCLUSION

The CALO project presents a combined challenge, being both a research effort into intelligent, adaptive, personal assistants and a large-scale deployable software system. This paper has described the PTIME system, the Personalized Time Manager of a CALO agent. PTIME is designed to assist a human user in managing her time commitments, a large part of which involves managing the communication and negotiation intrinsic in scheduling meetings. We have presented the motivations behind our design decisions, the deployment challenges we faced, the issues that arose with actual usage, and our current work on addressing them.

Our experience in developing PTIME has led to the following lessons that now inform our current research and development:

- *Scheduling technology is only one part of a personalized time management solution*. To be useful, an assistive agent must address issues of privacy and authority while providing support in solving the user's most important problems. To be usable, the agent must be deployed in the real world, an open environment not fully under the user's or agent's control.
- *Multiple modes of communication and operation are critical to successful scheduling*. An open system must be able to function within the real world. For a scheduling system, this means the ability to communicate and negotiate with other agents, be they CALOs with complementary PTIME components, CALOs or human users with email, or human users with voicemail. Crucially, it also means being able to function in spite of incomplete or uncertain information.
- *A well-designed user interface is key not only to making a system usable but also to promoting the use of new functionality*. Design flaws of the initial user interface hindered adoption and masked the full set of PTIME capabilities. Strong usability allows users to explore advanced capabilities and leads to more frequent use of the overall technology.
- *A personal assistant must build trust*. Until users understand the benefits of a new technology, they will be reluctant to abandon their tried-and-true practices. An assistant can facilitate this transition by providing users with visibility into its reasoning process—for example, by providing progress reports, and explanations, especially of problems or failures.
- *An assistive agent must aim to support, rather than replace, the user's natural processes*. Since users are often unwilling to relinquish full control of their time management, an assistive agent must be designed primarily to facilitate rather than automate the user's scheduling processes. This means implementing a process that closely follows the user's own practices (including her evaluation criteria for a candidate schedule) and making explicit the various decision points to support adjustable autonomy.

Acknowledgments. We thank all the members of the CALO project, especially our collaborators at SRI, University of Michigan, and USC. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

8. REFERENCES

[1] D. Azuma. Glow API version 1.0 reference. July 2000.

[2] P. Berry, M. Gervasio, B. Peintner, T. Uribe, and N. Yorke-Smith. Multi-criteria evaluation in user-centric distributed scheduling agents. In *Proc. of the AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*, Mar. 2006.

[3] P. Berry, M. Gervasio, T. E. Uribe, and N. Yorke-Smith. Mixed-initiative issues for a personalized time management assistant. In *Proc. of ICAPS'05 Workshop on Mixed-Initiative Planning and Scheduling*, pages 12–17, Monterey, CA, June 2005.

[4] A. Cheyer and D. Martin. The Open Agent Architecture. *J. Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, 2001.

[5] A. Cheyer, J. Park, and R. Giuli. IRIS: Integrate. Relate. Infer. Share. In *Proc. of Fourth Intl. Semantic Web Conference Workshop on the Semantic Desktop*, Galway, Ireland, Nov. 2005.

[6] A. H. W. Chun, D. W. M. Yeung, G. P. S. Lam, D. Lai, R. Keefe, J. Lam, and H. Chan. Scheduling engineering works for the MTR Corporation in Hong Kong. In *Proc. of AAAI-05*, 2005.

[7] F. Dawson, D. Stenerson, and E. H. Durfee. RFC:2445. *Internet Engineering Task Force, Network Working Group*, 1998.

[8] E. Ephrati, G. Zlotkin, and J. Rosenschein. A non-manipulable meeting scheduling system. In *Proc. of the Thirteenth Intl. Distributed Artificial Intelligence Workshop*, Seattle, WA, 1994.

[9] A. Faulring and B. A. Myers. Enabling rich human-agent interaction for a calendar scheduling agent. In *Proc. of CHI-05*, 2005.

[10] M. S. Franzin, F. Rossi, E. C. Freuder, and R. Wallace. Multi-agent constraint systems with preferences: Efficiency, solution quality, and privacy loss. *Computational Intelligence*, 20:264–286, May 2004.

[11] M. T. Gervasio, M. D. Moffitt, M. E. Pollack, J. M. Taylor, and T. E. Uribe. Active preference learning for personalized calendar scheduling assistance. In *Proc. of IUI'05*, San Diego, CA, Jan. 2005.

[12] N. R. Jennings and A. J. Jackson. Agent-based meeting scheduling: A design and implementation. *IEE Electronics Letters Journal*, 31(5):350–352, 1995.

[13] T. Joachims. Making large-scale SVM learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*. MIT Press, 1999.

[14] E. Junker. QuickXplain: Preferred explanations and relaxations for over-constrained problems. In *Proc. of AAAI-04*, 2004.

[15] W. Mark and R. Perrault. CALO: Cognitive Assistant that Learns and Organizes. `www.ai.sri.com/project/CALO`, 2005.

[16] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1–2):149–180, 2005.

[17] P. J. Modi, M. M. Veloso, S. F. Smith, and J. Oh. CMRadar: A personal assistant agent for calendar management. In *Proc. of Agent-Oriented Information Systems (AOIS 2004)*, LNCS 3508, pages 169–181, Riga, Latvia, June 2004.

[18] M. D. Moffitt, B. Peintner, and M. E. Pollack. Augmenting disjunctive temporal problems with finite-domain constraints. In *Proc. of AAAI-05*, pages 1187–1192, Pittsburgh, PA, July 2005.

[19] D. Morley and K. Myers. The SPARK agent framework. In *Proc. of AAMAS'04*, pages 714–721, New York, NY, July 2004.

[20] K. L. Myers and D. N. Morley. Human directability of agents. In *Proc. of First Intl. Conf. on Knowledge Capture*, Victoria, BC, 2001.

[21] J. Oh and S. F. Smith. Learning calendar scheduling preferences in hierarchical organizations. In *CP'04 Workshop on Preferences and Soft Constraints (Soft'04)*, Toronto, Canada, Sept. 2004.

[22] L. Palen. Social, individual and technological issues for groupware calendar systems. In *Proc. of CHI-99*, pages 17–24, 1999.

[23] R. Payne, R. Singh, and K. Sycara. Rcal: A case study on semantic web agents. In *Proc. of AAMAS'02*, pages 802–803, 2002.

[24] C. Rich and C. Sidner. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3/4):315–350, 1998.

[25] S. Sandip and E. Durfee. A formal study of distributed meeting scheduling. *J. Group Decision and Negotiation*, 7:265–298, 1998.

[26] H. M. Sheini, B. Peintner, K. A. Sakallah, and M. E. Pollack. On solving soft temporal constraints using SAT techniques. In *Proc. of CP'05*, pages 607–621, Sitges, Spain, Oct. 2005.

[27] SICS. *SICStus Prolog User Manual Version 3.12*, Oct. 2005.

[28] P. Viappiani, B. Faltings, V. S. Zuber, and P. Pu. Stimulating preference expression using suggestions. In *AAAI 2005 Fall Symposium on Mixed-Initiative Problem-Solving Assistants*, Arlington, VA, Nov. 2005.