Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

SIMULATING USERS TO SUPPORT THE DESIGN OF ACTIVITY MANAGEMENT SYSTEMS

Julie S. Weber
Martha E. Pollack

Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109, U.S.A.

ABSTRACT

We describe a simulation system that models the user of a calendar-management tool. The tool is intended to learn the user's scheduling preferences, and we employ the simulator to evaluate learning strategies. The simulated user is instantiated with a set of preferences over local and global features of a schedule such as the level of importance of a particular meeting and the amount of preparation time available before it is to begin. The system then processes a set of simulated meeting requests, and over time and through user feedback, it learns the user's preferences, affording it the ability to thereafter manage the user's schedule more autonomously.

1 INTRODUCTION

We are interested in the design of interactive activity management systems able to assist their users in carrying out their daily plans. There is a broad spectrum of users targeted by such systems, ranging from people with cognitive impairment, who use them to help remember routine daily activities (such as eating, drinking, taking medication, cleaning, etc.) to busy executives, who use them to keep track of a large set of meetings and a long to-do list. Examples of the former include *Autominder* (Pollack et al. 2003, Rudary, Singh, and Pollack 2004), COACH (Mihailidis, Fernie, and Barbenel 2001), the *Aware Home Project* (Abowd et al. 2000), and the *Assisted Cognition Project* (Kautz et al. 2002). Systems used for schedule and meeting management include CABINS (Miyashita and Sycara 1995), Carnegie Mellon's CMRadar (Modi et al. 2004), and SRI's *Personalized Time Management Assistant* (Berry et al. 2004).

Designing activity management systems presents significant challenges, in part because of the characteristics of the intended users. People with cognitive impairment are unlikely to be able to handle "alpha"-version software as they may become confused in the face of software errors, while busy executives are unlikely to be willing to use "in-development" systems and will instead demand robust, off-the-shelf products. Thus, there is an important role for systems that simulate the users targeted by activity management systems. By simulating users and their environments, we can test alternative designs and approaches to activity management before sending the systems to actual users.

Here we present a case study of a user simulation that we have designed and tested to investigate the feasibility of employing machine-learning techniques to infer meeting-scheduling preferences of the user of an activity management system. In Section 2 we discuss other research similar to our own. Section 3 describes the target activity management system that we have investigated using our simulated user, followed by a description of the simulated system itself in Section 4. Section 5 reports on experiments we conducted using the simulator. Section 6 presents a discussion of future work and draws conclusions.

2 RELATED RESEARCH

Simulating human behavior is a challenging task, on which a great deal of work has been done; see Giordano et al. (2004) for an excellent overview of the field. Of particular relevance to our work are systems that simulate the users of computer systems, for the purpose of facilitating the development and testing of those systems. Typical examples of such work include the simulation of cell-phone users to predict the effects of alternative interfaces on driving behavior (Salvucci 2001) and the simulation of air-traffic controllers to understand the trade-offs in different ATC designs (Gluck and Pew 2001). Also of interest are applications to computer security: for example, Putezka et al. (1996) simulate both authorized and malicious users of computer systems, in order to create data that can be used in studying the detection of security breaches. Boddy et al. (2005) take a similar track, using AI planning methods to generate adversarial courses of action that a simulated user of a computer system might take.

A recent effort to model the users of activity management systems was done by Rudary, Singh, and Pollack (2004) who constructed a simulator for a memory-impaired user of the Autominder cognitive orthotic system. As in our current project, they were interested in using this simulator to study the feasibiliy of adding machine learning to activity management; however, there are several differences with our own work, including the fact that they were specifically modeling an impaired user and they were learning reminder strategies as opposed to scheduling preferences. Gervasio et al. (2005) also constructed a simulator for the user of an activity-management system–in fact, the same activity-management system that we target; they used this system to study the effects of alternative active-learning strategies to support an underlying Support Vector Machine that was tasked with learning a user's scheduling preferences. While that work is a direct precursor of what we report on here, our extensions include a richer set of user-feedback options, and a focus on learning whether and when the system should act autonomously on its user's behalf.

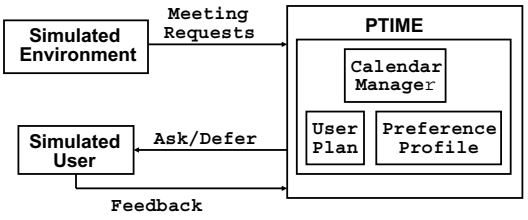

Figure 1: System Architecture

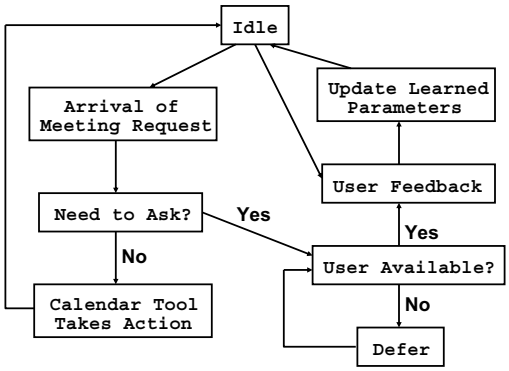

Figure 2: System Flow Diagram

3 THE TARGET SYSTEM

The simulation system that we describe in this paper is intended to model users of the PTIME activity-management system. PTIME, or the Personalized Time Manager (Berry et al. 2005), is designed to help a user manage his or her calendar, unobtrusively learning user preferences (e.g., about when meetings should be scheduled) using passive learning, active learning, and explicit advice-taking, and, over time, becoming more and more autonomous. The system maintains a model of the user's plan–i.e., the things that he or she must do and constraints on how and when they should be done–as well as a user preference profile. Note that PTIME is capable of handling both meetings with fixed times (e.g., next Tuesday from 4 to 5pm), but also "to-do" items with floating times (e.g., for about an hour, sometime before next Friday). However, in this paper we focus on fixed meetings.

Figure 1 shows how PTIME is integrated with our simulator. Note that we simulate both the user and the user's environment, where the latter comprises the set of meeting requests received each day. Figure 2 shows the flow of information through the system. First a meeting request arrives and is processed by the calendar tool, resulting in one of three outcomes: it is accepted and consequently added to the user's plan, it is rejected, or a query is generated so that the user can decide what to do with the request. In the case in which the user must be queried, the system must make an additional decision: can the user currently be interrupted? If so, the query will be issued immediately (and the expectation is that the user will reply immediately); if not, the query will be deferred until the user is available.

In addition to interactions triggered by meeting requests, the user can at any time provide feedback to the system, alerting it to its mistakes and providing a metric for its level of performance. We differentiate between two forms of user feedback. One type of feedback involves providing the calendar tool with a numeric score indicating the number of correct decisions (accept or reject) that have been made over a given amount of time (a day, a week, etc.). With the other form of feedback, the user instead identifies the specific decisions that were incorrect. Note that while this requires the input of more information, it may actually be more natural for the user. In fact, it may not be necessary to explicitly request such information; instead, it can be obtained by observing the user as he or she makes corrections to the calendar to undo actions taken by the system: moving or deleting meetings that were scheduled, and adding meetings that were rejected. (For this to work, the user must have access to a list of rejected requests.) With either form of feedback, the system then adjusts its user preference profile.

Our research goal is to address the problem of enabling a calendar tool to learn its user's preferences with respect to meeting-scheduling, so as to make appropriate decisions about how to deal with meeting requests as they arrive. In particular, a short-term subgoal is to compare the usefulness of the two alternative methods of user feedback, to give us a sense of how long it takes for the system to converge to correct preference profiles, given different amounts of information about performance. The user simulator is designed to help us achieve these goals as efficiently as possible, by conducting many more tests than would be feasible with human users.

4 USER SIMULATOR

As noted in the previous section, we simulate both the user and his or her environment; in this section we discuss each of these in turn.

4.1 User Model

The most important aspect of the user that we need to model is his or her meeting preferences; in particular we model three types of preferences:

1. Preferences over times of day and days of the week for meetings, e.g., does the user prefer morning or afternoon meetings?
2. Preferences with respect to the arrangement of free time in the schedule during the week, e.g., does the user prefer meetings clustered together, with relatively large blocks of free time, or does s/he prefer meetings more evenly interspersed throughout the week?
3. Preferences with respect to the amount of preparation time needed for each meeting, e.g., how much advance notice does the user require for a meeting?

To model the first type of preference, we partition the work week into 25 blocks, 5 per day, corresponding to early morning, late morning, lunchtime, early afternoon and late afternoon, and each block of time is associated with a preference value from 1 to 4, with 1 representing a maximally preferred slot and 4 representing a minimally preferred slot. Every meeting is assigned an importance value, and to schedule a meeting in a particular time block, its importance must exceed the preference level assigned to that block. (Learning importance values from meeting subjects and participants is a topic for future research.)

To model the second type of user preference, we again use a 4-valued preference in the user model. Some users may prefer long blocks of uninterrupted free time while others may be more interested in a large number of shorter blocks of free time. A user who prefers to have meetings broadly dispersed is assigned a value of 1; a value of 2 is used to model a user who has a preference for some large blocks of free time, but still values dispersal of meetings more highly; a value of 3 models the reverse–i.e., a user who wants blocks of free time, but still doesn't want an excessive amount of clustering of meetings; and a value of 4 models a user whose main preference with respect to clustering is to have meetings clustered as much as possible, hence preserving blocks of contiguous free time that are as long as possible. More specifically, a simulated user with less than one-quarter of its meetings scheduled within a block of two or more meetings is assigned a value of 1, whereas a user whose meetings are clustered more than three-quarters of the time is assigned a 4.

The third user characteristic is his or her preference for meeting-preparation time. If a meeting request is sent at noon for a 1pm meeting, the user may not have sufficient time to prepare, and so might prefer to defer the meeting or even reject it outright. In the current simulation, we make the admittedly unrealistic assumption that preparation time is independent of meeting type and importance. We again model each user with one of four values, where 1 represents a preference for less than 30 minutes of preparation time, and 2, 3, and 4 represent preferences for between 30 and 60, 60 and 120, and more than 120 minutes of preparation time respectively.

When we simulate a user to conduct testing, we begin by setting the values of these three preference features. The calendar tool is also instantiated with an initial preference profile, which may be randomly generated or may be created by making an inference from the entries already in the user's plan.

During experimentation, the activity-manager is presented with a meeting request and must make a decision about whether to accept it, reject it, or query the user about what to do. For this purpose, it consults its current preference profile. In the experiments reported on in the next section, we make use of a very simple method for comparing the current request with the preference profile, namely, we require that all three thresholds established by the profile be exceeded. In other words, in order to be accepted, a meeting request for a given time slot T must (1) have importance greater than or equal to preference value associated with T, (2) not result in the maximum percentage of unclustered meetings within the user's schedule being exceeded *and* (3) have arrived early enough to permit the amount of preparation time required by the preparation-time value. If any of these three conditions are violated, the system rejects the meeting request. If all three are satisfied, and, in addition, the meeting request does not conflict with an already scheduled meeting, then the system accepts the request. Otherwise (if the conditions are satisfied but there is a conflict), then the system queries the user about what to do. The simulated user then schedules or rejects the meeting in accordance with the fixed set of preferences with which the user was instantiated.

The effect of combining preferences in this way is that we actually view the preferences as requirements: they all must be satisfied for a meeting to be scheduled. A future goal is to modify the combination function, to employ a weighted sum function instead.

The key role for the user simulation in our experiments is to generate feedback to the system. It does this by considering each meeting request itself and using its set of preferences to decide whether or not the meeting should be scheduled. The calendar tool is considered to have made

a mistake whenever the action it chooses does not match with the one the simulated user believes to be correct; such a mistake represents an error in the preference profile, which must be adjusted by the calendar tool. As described above, mechanisms for user feedback include periodically providing the calendar tool with a score representing the number of correct decisions made, or implicitly scoring the system by manually adjusting meetings already accepted or rejected by the system.

4.2 Model of User Environment

The simulated environment comprises the meeting requests that arrive each day for processing. Currently we use a very simple model in which, n times each day, the user receives a meeting request to be scheduled, where the day, time, and importance of the meeting are all randomly generated. We choose n as an approximation of the number of meeting requests that an arbitrary user might receive on a given day, noting that this parameter can be set and adjusted during simulation.

5 INITIAL EXPERIMENTS

In a preliminary series of experiments, we used our user simulator to study the feasibility of a particular approach to preference learning: a limited version of hill-climbing over the space of preference profiles. The system receives meeting requests and makes decisions about what to do, and then periodically receives feedback from the user, as described in the previous section. We refer to the period of time between the receipt of feedback as a *training episode*. At the end of each training episode, when feedback is received, the system needs to adjust its current user preference profile. To do this, it first compares its performance in the current episode against a stored preference profile, which represents the best one found so far. Performance is measured as the percentage of correct decisions made during an episode: the number of correctly accepted and correctly rejected meetings divided by the total number of meeting requests. (Thus, queries to the user do not improve the score.) The better of the two candidate preference profiles–the current one or the stored one–is selected, and becomes the new stored profile. In addition, the selected one is modified, as explained below, to create the preference profile for the next training episode.

The form of the modification depends upon the type of feedback received from the user. When the user simply provides an overall score, the system randomly adjusts each of its three preference values: each value is either increased by one, decreased by one, or left unchanged, with probability 1/3 each. (If the value before modification was 1 (4), then with probability 1/3 it is adjusted up (down), and otherwise it remains unchanged.)

With the more informative feedback mechanism, the update process is more principled. In particular, given the nature of the preference combination function, a great deal can be learned when the user adds a meeting that was initially rejected by the calendar tool. This is because, when it rejects a meeting request, the calendar tool can readily keep track of its reason(s) for doing so, i.e., it records which of the three preference values were not satisfied. So, for example, if a meeting was rejected because it was not important enough for its time slot, the stored reason would be *importance*; if it was rejected because it caused the user's schedule to become too crowded with singleton meetings and because there was not enough preparation time for it, the stored reasons would be *free-time, prep*. If a rejected meeting is later restored by the user, then the current values for *all* the reasons associated with its rejection must be too high, and consequently they are each immediately lowered. Then, when a new preference profile is generated, the current values serve as upper bounds on the new values.

Unfortunately, it is not as easy to learn from the case in which the user rejects a meeting that the calendar tool accepted. Because of the structure of the preference rule, in which all three preferences must be satisfied, it is impossible to know which of the preference values needs to be raised in this situation: it could be any or all of them. The system thus currently does not use these instances in generating a new preference profile, but reverts to a random change in the preference values for features for which the only feedback is the rejection of an initially accepted meeting.

5.1 Preliminary Experimental Results

In order to test our learning mechanisms, we implemented our simulator in Java and ran it on a sparcv9 at 1062MHz. We conducted experiments in which we first randomly generated a set of preference values for a simulated user, and populated the user's calendar using these preferences to bias meeting placement. We also randomly (and independently) generated an initial preference profile for the calendar tool to use. We then conducted two runs, one for each of the feedback mechanisms. In each run, we generated 20 fixed-time meeting requests per day, and had the simulated user provide feedback at the conclusion of each day, and we allowed the system to run until its learned preference profile was exactly equivalent to the preferences of the simulated user. We repeated this process for 20 different simulated users (i.e., 20 sets of preference values) for a total of 40 runs.

The graph in Figure 3 depicts the results of our initial experimentation, smoothed for readability and cut off after 1000 days of processing. (We used a 4253H smooth. The average number of mistakes per day with pure scoring dropped to exactly zero close to day 3000.)

The x-axis measures the day of the simulation and the y-axis represents the number of errors per day; the results

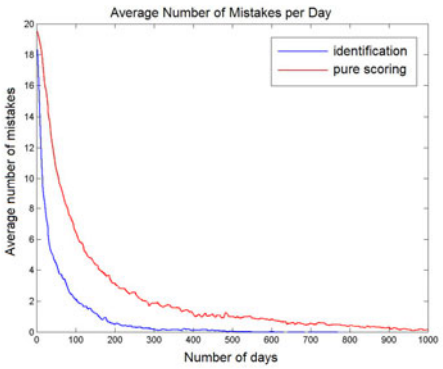

Figure 3: Comparison of Feedback Mechanisms

are averaged over each of the 20 runs. The two lines on the graph show the results for each of the two feedback mechanisms.

Even though this is just a preliminary experiment, we can draw two conclusions. First, and not surprisingly, the experiment confirms our hypothesis that it is worth using a feedback mechanism in which mistakes are explicitly identified: the average number of errors per day is always smaller with this method than with the summary-scoring method. Second, while convergence to an exact preference model, even with the explicit feedback method, takes an inordinately long time (approximately 500 days), it takes less than 30 days to achieve a level of performance with 25% or fewer errors per day; this is approaching a duration that may make the approach feasible for real applications.

6 FUTURE WORK AND CONCLUSIONS

The goal of our research is to create a computational agent capable of managing the schedule of its user, learning the user's preferences so as to become more autonomous over time. Testing such a system, and comparing alternative learning procedures, alternative mechanisms for user feedback, and so on, can be very time consuming. With a simulated user, we can relatively quickly test many alternatives without a human-in-the-loop. To date our simulated user is quite simple, consisting mainly of a model of preferences used to provide feedback to the system, and the experiments we conducted to date have been quite preliminary. Even so, they suggest that it may be feasible to use an explicit feedback mechanism to converge to a reasonably accurate preference profile within an reasonably small amount of time. Of course, there are some things that cannot be tested with a simulated user: for example, there is no way to determine which of the alternative feedback methods is more attractive to a real user.

There are a number of ways in which our current system can and should be expanded. First, and most obviously, there are other user features that should be modeled–and there are also additional features of the meetings themselves that our calendar tool should take into account in deciding how to process a meeting request. For example, it should consider how hard it will be to cancel a meeting once accepted, and it should consider what the opportunity-cost will be for rejecting or deferring a meeting request. At present, we assume that there is essentially no cost to canceling an accepted meeting, and that an initially rejected (or deferred) meeting can always be scheduled later, but of course both of these assumptions are too strong. Second, as we mentioned earlier, we make use of a very simple function for combining preferences in deciding how to process a meeting request, and a more realistic model would use a richer aggregation function, such as a weighted sum. Third, we have so far only considered meetings with fixed times, but we need also to consider floating meetings and other to-do tasks. Fourth, we have not yet properly handled requests for meetings that conflict with something already scheduled; in the current version of the system, we simply ask the user what to do in such cases. Instead, we would like to enable the system to decide when to schedule a conflicting meeting (and likewise whether to cancel the meeting with which it conflicts or leave it on the schedule as well), reject a conflicting meeting, or resort to asking the user or deferring the request.

ACKNOWLEDGMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Dept. of the Interior, NBC, Acquisition Services Division, under Contract No. NBCH-D-03-0010. 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).

REFERENCES

Abowd, G. D., A. F. Bobick, I. A. Essa, E. D. Mynatt, W. A. Rogers. 2002. The Aware Home: Developing Technologies for Successful Aging. In *Proceedings of the AAAI-02 Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care*, 1-7.

Berry, P. M., M. Gervasio, T. E. Uribe, M. E. Pollack, M. D. Moffitt. 2005. A Personalized Time Management Assistant. *AAAI Spring Symposium*, 20-27.

Boddy, M., J. Gohde, T. Haigh, and S. Harp. 2005. Course of Action Generation for Cyber Security using Classical Planning. In *Proceedings of the 15th International Conference on AI Planning and Scheduling*.

Gervasio, M. T., M. D. Moffitt, M. E. Pollack, J. M. Taylor, T. E. Uribe. 2005. Active Preference Learning for Personalized Calendar Scheduling Assistance. *International Conference on Intelligent User Interfaces*, 20-27.

Gluck, K. A., and R. W. Pew. 2001. Overview of the Agent-Based Modeling and Behavior Repreesntation (AMBR) Model Comparision Project. In *Proceedings of the 10th Conference on Computer Generated Forces and Behavior Representation*, 3-6.

Giordano, J. C. 2005. Exploring the Constraints of Human Behavior Representation. In *Proceedings of the 2004 Winter Simulation Conference*, Volume 1.

Kautz, H., L. Arnstein, G. Borriello, O. Etzioni, D. Fox. 2002. An overview of the assisted cognition project. In *Proceedings of the AAAI-02 Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care*, 60-65.

Mihailidis, A., G. Fernie, J. C. Barbenel. 2001. The Use of Artificial Intelligence in the Design of an Intelligent Cognitive Orthosis for People with Dementia. *Assistive Technology*, 13:23-39.

Miyashita, K. and K. Sycara. 1995. CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair. *Artificial Intelligence*, 76:377-426.

Modi, P. J., M. Veloso, S. Smith, J. Oh. 2004. CMRadar: A Personal Assistant Agent for Calendar Management. In *Sixth International Workshop on Agent-Oriented Information Systems*, 134-148.

Pollack, M. E., L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos. 2003. Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment. *Robotics and Autonomous Systems*, 44:273-282.

Puketza, N., K. Zhang, M. Chung, B. Mukherjee, R. Olsson. 1996. A Methodology for Testing Intrusion Detection Systems. *IEEE Transactions on Software Engineering* 22 (10) 719-729.

Rudary, M., S. Singh, and M. E. Pollack. 2004. Adaptive Cognitive Orthotics: Combining Reinforcement Learning and Constraint-Based Temporal Reasoning. *Twenty-first International Conference on Machine Learning*, 719-726.

Salvucci, D. D. 2001. Predicting the Effects of In-Car Interfaces on Driver Behavior Using a Cognitive Architecture. In *Proceedings of the SIG-CHI Conference on Human Factors in Computer Systems*, 102-127.

AUTHOR BIOGRAPHIES

JULIE S. WEBER is a PhD student at the University of Michigan. She was a Dean's Fellow at Tufts University where she received her Masters Degree in Computer Science in 2004, and she received her Bachelors Degree in Computer Science and Mathematics from Wellesley College in 2003. Her e-mail address is <weberjs@umich.edu>.

MARTHA E. POLLACK is Professor and Associate Chair for Computer Science and Engineering in the Department of Electrical Engineering and Computer Science at the University of Michigan in Ann Arbor. From 1985 to 1991, she worked at the AI Center at SRI International. She was a faculty member in the Department of Computer Science at the University of Pittsburgh from 1991 to 2000, where she also served as Director of the Intelligent Systems Program from 1998 to 2000. She began her current position at Michigan in Sept., 2000. Pollack is a Fellow of the American Association for Artificial Intelligence, and a recipient of the Computers and Thought Award (1991), an NSF Young Investigator's Award (1992), and the Univ. of Pittsburgh Chancellor's Distinguished Research Award (2000). Her e-mail address is <pollackm@umich.edu>, and her web page is <www.eecs.umich.edu/$\sim$pollackm>.