23rd Annual Conference of the German Association of Computer Science (Gesellschaft für Informatik, GI), Dresden, Germany, Sept. 93

Office Automation Systems that are "Programmed" by their Users

Siegfried Bocionek[1] and Tom M. Mitchell[2]

[1] Siemens AG, ZFE ST SN 33, Otto–Hahn–Ring 6, 8000 München 83, Germany
[2] Carnegie Mellon University, School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891, USA

Abstract. To be highly useful, office automation systems require customization to individual users, work environments, and tasks. We consider the question of whether office automation systems can be developed that allow users who are not skilled programmers to easily "program", or customize, these tools themselves. In other words, can we deploy very general, non-customized programs that can be extended and adapted during usage? Besides the well-known *knowledge acquisition* and *automated learning* methods we propose an approach called *dialog-based learning* (DBL), that allows the user to teach the system directly while performing the task. It acquires knowledge through a dialog in which the user both *illustrates* the procedure using a grounded example, and *provides instructions* about how to perform the task in the general case. We focus discussion on CAP II (a program that schedules meetings by negotiation via email), and on RAP (a program that makes room reservation via email).

1 Introduction

We are interested in office automation systems that one might call *software secretaries*: software agents that provide assistance in managing work-related activities (e.g., reserving conference rooms, processing purchase orders), in much the same way as human secretaries. This concept seems attractive, since such software could make available to everyone within an organization secretarial assistance that is presently available only to a small fraction of workers.

The primary impediment to achieving widespread, inexpensive software secretaries is the software development, customization, and maintenance cost involved. Human secretaries typically require months in a new job before acquiring the varied types of necessary knowledge. Software secretarial assistants will have to be customized in the same way. The cost of (traditional) customization, however, is a major obstacle to the widespread appearance of software secretarial assistants.

One way to dramatically reduce these software costs would be to develop a single *user-independent* software secretary, and to endow it with features that allow it to be trained in the field. This paper considers the question of how to organize such field-customizable systems, if their users are not skilled programmers. Our work on the calendar assistant CAP [3] has shown that machine learning techniques can play a role in providing the kind of user customization desired. It has also shown the limitations of such fully automatic learning methods. This paper describes a successor system, CAP II, as well as a room reservation apprentice, RAP, which are currently being developed. Both take a broader approach to customizing software in the field, by including a semi-automated knowledge acquistition technique called *dialog-based learning (DBL)* that engages the user in a dialog with the system to teach it office procedures.

There is good deal of related work in calendar and other workflow applications. Medina-Mora et al. [6] have proposed a general workflow model that fits many office tasks. A procedure for using contract net protocols to schedule meetings was reported by Sen and Durfee [8]. The communicating automata of CAP II are an extension to that approach. Another implementation of meeting negotiation through contract nets was done by Lux [5]. Based on a general "negotiation language", calendar agents were specified and interfaced to existing calendar tools (e.g., Xcalentool). While the above approaches have no learning capabilities, the negotiating calendar of Kozierok and Maes [4] does. The difference from our CAP II system in learning is that it uses mainly memory-based learning, while we explicitly construct general rules. Nakauchi et al. [7] have developed a related learning system for the processing of purchase orders. Their system learns rules that help the user complete the purchase order form.

2 CAP II: A learning, communicating, and negotiating Calendar APprentice

The calendar *apprentice* CAP [3] is a calendar program that learns to give advice to its user on how to set up a meeting (i.e. how long, where, when). The advice can be accepted or overridden by the user, and is generated by rules that are inductively learned from previous events that the user has entered on the calendar. E.g., the rule

```
IF AttendeeStatus = Undergrad
AND AttendeeDepartment = ComputerScience
THEN duration = 30
```

predicts that meetings with computer science undergraduate students should last 30 minutes. By applying its learning programs automatically each night to the most recently added calendar events (ID3 and Backpropagation are used), CAP learns rules such as the above, that characterize individual user preferences.

CAP II extends CAP by providing facilities that support autonomous negotiation of meeting dates via email. Once the user has indicated the desire for a meeting, CAP II generates a proposal and sends it to all attendees. If all recipients possess CAP II, their agents may negotiate until a commonly acceptable date is found. If one recipient does not have CAP II, the message is human readable and he may respond directly.

A collection of CAP II agents (see also [1]) forms a system of communicating finite state machines (FSM) that interact by means of a contract-net protocol similar to that of Sen and Durfee [8]. The involvement of non-CAP users is ensured by a readable, semi-structured email message format and some special supporting mechanisms such as suggesting at the end of each message how to reply. An example of an agent-generated email message (although for the RAP domain) is shown in Fig. 1.

```
To:main.office@cs
From:jean@cs
Subject:Need a room on dec-24-1992, 2:30pm RAP-Requestxx123

I need a lecture room.
Please let me know if there is one available as follows.

  Date: dec-24-1992
  Time: 2:30 pm
  Duration: 90 min
  Seats: 30 - 50
  Speaker: M. Sassin, MIT

You can make your reply computer-understandable by
beginning your message with one of the following:
"No" or a room number
```

Fig. 1. A request for a lecture room as generated by a RAP agent

In addition to learning rules characterizing users' preferences CAP and CAP II contain further knowledge-acquisition mechanisms. One example in CAP is the collection of data about persons when they are mentioned to the system the first time. This data, e.g. position and affiliation, provides the feature values refered to by the learned rules. Additional knowledge acquisition in CAP II leads to the construction of a global alias list of user addresses. Since all acquisition works through questions that are asynchronously initiated by the agent, we consider this a simple application of dialog-based learning.

CAP and CAP II are examples of systems that increase their usefulness during operation. By learning the users' scheduling preferences they continuously reduce the effort needed by the user to schedule meetings. By autonomous negotiation they reduce the number of computer interactions (or analogous phone calls). The key point is that the user *himself* provides the system with the examples necessary to generate this additional functionality. However, he does not need to write code; just *using* the tool is sufficient.

3 RAP: (Proposal for) A Room APprentice that is "Completely" Programmed through its Usage

RAP (Room APprentice) is intended to be an agent that assists the user in reserving meeting and lecture rooms. It is a case study in the use of dialog-based learning to acquire general workflow procedures. In particular, RAP acquires new procedures for reserving rooms by watching the user during his regular email interaction, and requesting from the user details about the contents and semantics of the messages sent and received.

Table 1. RAP's empty FSM

	ϵ	stop-RAP
global-start	–	goto global-stop
global-stop	clean-up process-exit	–

RAP models the way room reservations may be handled in a typical organization. A person who wishes to reserve a room first sends an email request to a particular secretary who administers rooms. If the response is negative, one might then send the same request to another secretary responsible for a different set of rooms, and so on. The task of each RAP agent is to learn to perform the complete search for a room automatically, on behalf of its user. RAP composes, sends and receives email messages until a room is found (or can't), then presents the result to the user.

RAP is different from the calendar system CAP II in several ways. First, it copes with a different domain. Second, RAP does not give the user advice in the sense of CAP II. However, RAP is similar to CAP II in performing automated negotiations via email messages, although the negotiation of a RAP agent is simpler. The major difference is, however, that RAP is intended to synthesize its *operational structure* from scratch. The complete finite state machine (FSM) that drives the negotiation is *learned* during the operation of RAP by analyzing email messages which were sent and received by the user. The basis for this learning is our DBL approach.

Some task-independent rudimentary structure is pre-programmed into RAP. The specific RAP messages are filtered by means of keywords in the subject line (the word `"RAP-Request"` in Fig. 1 is used for filtering, the connected word `"xx123"` for identifying a specific event; those mechanisms work exactly like in CAP II [1]).

The pre-programmed behavior of RAP includes primarily the "empty" FSM that contains only start and stop states, and the "meta format" of the email messages. The strategy for adding states and transitions to the FSM is fixed.

Sending email and notifying the user are the major actions.[3]

Table 1 shows the "empty" FSM with which RAP is initialized, with the states as indices of the rows and the vocabulary as indices of the columns. The only phrase that may be received by the "empty" FSM is *stop-RAP* which causes RAP to go to state `global-stop` and exit. In order to synthesize the automaton that characterizes the room reservation method, RAP has to learn the *vocabulary* (i.e. the message types) and the necessary *states*. In the case of learning the vocabulary, DBL will work as illustrated in the following example. Suppose the user sends an email request to the room administrator in the main office as shown in Fig. 1. RAP analyzes the message *before* sending it. It asks the user the following questions:

```
To which type of interaction does this message belong?
(room-request, positive-answer, counter-proposal, other)
```

Assume the user answers with `"room-request"` (under some "good-will assumption"; see [1]). Then RAP will next ask

```
Which is the meaning-carrying phrase in that message?
```

Table 2. RAP's FSM after the learning of requests and positive answers

	ϵ	stop-RAP	room-request	positive-answer
global-start	–	goto global-stop	send-email goto s1	
global-stop	clean-up process-exit	–	–	–
s1	–	goto global-stop		notify-user(success) goto global-start

The user might answer `"need a room"` or only `"need"`. This answer is added to the set of phrases that are possible for a room request, and will be used to identify the types of future messages received by RAP. If the user answers the first question with `"other"`, RAP has to learn a *new type* as well. From the answers to the above questions the RAP agent builds up a thesaurus of the synonyms allowed for each message type.

If the FSM does not yet contain the message type as selected by the user, a *new column* has to be added to its transition table. Sending the analyzed email becomes the action of the entry FSM[state, new-message-type]. To determine the transition, the user will be asked if this action was the last one of the event. If yes, the transition is `goto global-start`. Otherwise, a new state is generated and added as a new row.[4] For example, RAP has learned to go to state s1 (see

[3] A discussion of the limitations and assumptions made in RAP can be found in [2].

[4] This strategy does not construct the minimal FSM because no merging of states happens. How to provide such an optimization is still under consideration (for details refer to [2]).

Table 2) *always* when the user sends a request with keyphrase `"need a room"`, and if `"RAP-Request"` appears in the subject line of an outgoing email message.

Table 2 shows the FSM after also having learned the message type **`positive-answer`**. The contents of the empty field FSM[global-start, positive-answer] will be synthesized only if and when that case occurs, i.e. when a positive answer for a request is received "before" sending the request (which would become an error case).

4 Conclusion and Future Work

This paper has explored the feasibility of creating user-independent office automation software that can be easily customized in the field. We have presented two case studies based on systems currently under development: CAP II for calendar management, and RAP for room reservation tasks. Both of them are intended to evolve to more effective, user-customized systems, by acquiring specialized knowledge about their particular user's preferences, task, and work environment. They contain three distinct learning mechanisms for acquiring user-specific knowledge:

1. *Knowledge acquisition* methods directly query the user for specific information on an as-needed basis, e.g. the position or email address of people.
2. *Automated learning* of general rules from specific actions taken by the user. For example, the rules learned by CAP to characterize user scheduling preferences.
3. *Dialog-based learning* acquires new workflow procedures by observing the user perform the task manually, and engaging the user in a dialog to extract the general procedure from the specific steps observed in this instance.

Dialog-based learning appears especially promising for field customization of interactive office systems, because it *makes use of the user* who is available while performing the task, to resolve the ambiguity inherent in inducing general procedures from specific traces. Thus one can say that the user himself "programs", because his interaction is the basis for the newly synthesized code.

If successful, the advantages to be gained from such field-customized systems are obvious. Office software need not be "complete" from the beginning; it can be extended incrementally during operation. This makes initial development faster and cheaper, and reduces the cost of maintaining code to accomodate new office procedures.

The long-term goal of this research is to produce a software secretarial agent that can assist with a wide variety of tasks, and can be easily field-customized to individual users. We plan to provide more secretarial tools that include additional email negotiation such as processing purchase orders. The increased diversity of tasks and users will allow for transfer of information acquired for one task to others, and will enable one agent to share what it has learned with other agents.

A second future direction involves interfacing our agents to existing information sources like the CMU online phone book and the UNIX `finger` command.

Interfacing to such pre-existing databases can significantly ease the problem of initially customizing to the work environment, as well as reducing the cost of updating the system over time (since those databases are already maintained by their providers). A third direction is to interface the system to an *electronic forms* system that allows communicating with remote users via structured pop-up menus, rather than text email messages.

Acknowledgments: We are grateful to all members of the Software Secretary group at CMU, who helped develop the programs and ideas presented here: Anand Chandani, Ratul Puri, Dan Ferrell, David Zabowski, Dayne Freitag, and Ryusuke Masuoka. This research was supported in part by Siemens, Digital, and a grant from ARPA.

References

1. S. Bocionek and T. Mitchell. CAP II: Making the Calendar Apprentice an Agent. Technical Report CMU-CS-93-to-appear, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1993.
2. S. Bocionek and M. Sassin. Dialog-Based Learning (DBL) for Adaptive Interface Agents and Programming-by-Demonstration Systems. Technical Report CMU-CS-93-to-appear, Carnegie Mellon University, School of CS, Pittsburgh, PA, 1993.
3. L. Dent, et al. A Personal Learning Apprentice. In *National Conference on Artificial Intelligence (AAAI-92)*, August 1992.
4. R. Kozierok and P. Maes. A learning interface agent for scheduling meetings. In *Int. Workshop on Intelligent User Interfaces*, Orlando, Fl, January 1993. ACM-SIGCHI.
5. A. Lux. A Multi-Agent Approach towards Group Scheduling. Research Report RR-92-41, DFKI, Kaiserslautern, Germany, August 1992.
6. R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The ActionWorkflow approach to workflow management technology. In J. Turner and R. Kraut, editors, *CSCW '92 (Sharing Perspectives)*, Toronta, Canada, November 1992. ACM Press.
7. Y. Nakauchi, et al. Groupware that Learns. In *Proc. of the IEEE Pacific Rim Communications, Computers and Signal Processing*. IEEE Press, May 1991.
8. S. Sen and E.H. Durfee. A formal study of distributed meeting scheduling: Preliminary results. In *Conference on Organizational Computing Systems*. ACM, November 1991.