On the design of a software secretary

Silvia Schiaffino* and Analía Amandi

ISISTAN Research Institute - Univ. Nac. del Centro Pcia. Bs. As.
Campus Universitario, Tandil, 7000, Bs. As, Argentina
{sschia,amandi}@exa.unicen.edu.ar
*also CONICET

Abstract Organizing our schedules is a common task that demands us time and effort. It would be helpful for us having a "software secretary" that could handle our calendars in a personalized way, as a human secretary does. Building such a software secretary is not a trivial task since it must have various capabilities: learning the user's preferences and habits, learning about the user's contacts and their preferences, deciding when and how to interact when its user in order to help him and executing actions autonomously to provide him assistance. These capabilities distinguish a software secretary from the current available computer programs that manage calendars. We propose in this work an architecture that enables agent developers to build software secretaries to assist users in different domains. We describe its architectural components and the interfaces between them in the context of calendar management.

1 Introduction

Managing our agenda and organizing our schedules are tasks we carry out almost every day, which demand us not only time but also effort. For example, to schedule a meeting with his clients a sales manager has first to determine a free time slot in his calendar, he has to choose a meeting place and he has to determine the duration of the meeting. Then, he has to find out if the meeting date and meeting time are convenient for his clients, and if they are not, he has to find another date, time and probably place. This manager also has to schedule meetings with his employees, meetings with his boss, gym classes twice a week, social dinners at the club, among other events. In order to schedule these events the sales manager considers, as everybody does, his priorities for different kinds of events, his preferences for meeting dates and times, his relationship with events' participants, his commitments and goals.

There exist some software packages that enable us to manage our calendars. Some of the various commercially available software for scheduling events are Calendar Manager [1], Office Tracker [2], MS Outlook and Visual Group Planner [3]. These systems enable users to add, modify and delete events to their schedules, and to visualize the information they have previously entered. However, our sales manager would have to do the same amount of work to manage his schedule since these systems do not provide him any help because they do

not know, and they cannot learn, his priorities and preferences for scheduling events.

There are some lucky people who have a secretary that can perform these scheduling tasks on their behalf. A secretary learns about his boss' scheduling preferences as she interacts with him daily. Thus, as time passes, she will know the kinds of events he usually attends, his scheduling priorities, she will know that he does not like meetings early in the morning, she will know his clients, his family and his friends. Thus, one can delegate the organization of the agenda to her.

It would be great for many of us to have a software secretary that could perform the same tasks a human secretary performs. Interface agents are a solution to this problem. Interface agents are computer programs that act as personal assistants to users with computer-based tasks. For example, an interface agent acting as a software secretary could help its user with his schedules in many different ways, such as suggesting meeting places, meeting times and meeting dates; notifying the user about incoming events; scheduling an appointment on the user's behalf; rejecting an invitation for a party because the user has a business dinner, and organizing a meeting with some customers, among other tasks human secretaries usually do.

Building such a software secretary as an interface agent is not a trivial task. In order to help its user this agent has to learn not only the user's preferences but also the tasks he usually performs (schedule events, cancel events), how the user performs these tasks (priorities, sequences of tasks), the relationships the user has with other people, how to interact with its user and how to deal with privacy concerns.

Several meeting scheduling agents have been developed in the last decade, but they have only partial assistance capabilities since they merely propose a user meeting dates, meeting times and meeting durations. Some of them are personal agents assisting a user in isolation, such as the ones presented in [12], [14] and [9]. Others are distributed negotiating meeting schedulers such as the ones proposed in [19], [7], [18], [20] and [16]. These agents do not consider many other important factors such as dependencies between these meeting characteristics and the priorities of a user's contacts. Others focus their attention in the negotiation process for scheduling a meeting, but they abandon the personalization aspects of this task. Other works, such as the one described in [5], focus their attention in the utilization of agent technology to support day-to-day activities (such as meeting scheduling) in human organizations.

In this work we propose an architecture that enables interface agent developers to build software secretaries in different application domains, which have the assistance capabilities mentioned above. Although several agent architectures have been proposed, such as [4], [6] and [15], they do not consider some components vital to the development of interface agents, mainly the user model and the learning component of this model. Thus, building software secretaries with these architectures would demand a lot of work. In this paper we describe the architectural components a software secretary should have and the interfaces

between these components. We describe how these components are materialized in the development of an agenda agent.

The paper is organized as follows. Section 2 describes the functionality of an agenda agent and the interactions that take place among a user and its agent. Section 3 presents the interface agent architecture we propose and how it is materialized in the context of calendar management. Finally, Section 4 presents our conclusions and future work.

2 An agenda agent

An agenda system enables a user to organize and visualize information about his activities. By using an agenda system a user can manage his calendar, i.e. he can schedule events, delete events, edit the information of an event, add priorities to events; visualize his calendar in different formats (daily, weekly, monthly); manage his contacts, i.e. add a new contact, edit the information of a contact, send an email to a contact and generate reports (a list of activities for a particular day). An event is typically described by many attributes: the event type (business meeting, gym class, a course, social meeting, appointment with the doctor, party), a description, a starting date, an ending date, a starting time, an ending time or a duration, a place in which it takes place, a list of participants (one or more), a state (confirmed, pending, canceled), a priority and a host. An event may be recurrent (daily, weekly, or monthly) or not. A user may also want to schedule activities for a particular day or for a week without specifying a time interval, such as: "go to the supermarket, pay the phone bill and go to the library".

An agenda agent will interact with both the agenda system and the user in order to assist him. An agenda agent can be viewed as a personal assistant or software secretary [13] that helps the user to cope with his calendar. A secretary can arrange meetings on our behalf, negotiate a meeting date and place with our clients, refuse an invitation for a party because we have a business dinner with our boss, suggest new dates to reschedule a meeting. In order to perform these tasks, an agenda agent will need to acquire the same knowledge a human secretary has about us, i.e. our preferences for meeting hours, meeting dates and meeting places, our priorities for scheduling business meetings, social events or classes, information about our contacts, such as the relationships we have with other people and their role in our life, and the activities we usually schedule. An agenda agent can learn about us in the same way a secretary does: by observing our behavior while we manage our calendar, through information we explicitly give her, and by the feedback we provide when she suggests us a meeting date or she cancels an event. Thus, once the agent has acquired enough knowledge, it can help us with our calendar management. In the same way we increase our trust in our secretary as we observe that she carries out well her tasks, we will increase our trust in our agenda agent as it shows an appropriate behavior. Depending on this trust we will enable the agent to perform or not some tasks

on our behalf. Figure 1 shows a general schema of a software secretary helping its user to manage his schedules. This figure is an adaptation from [17].

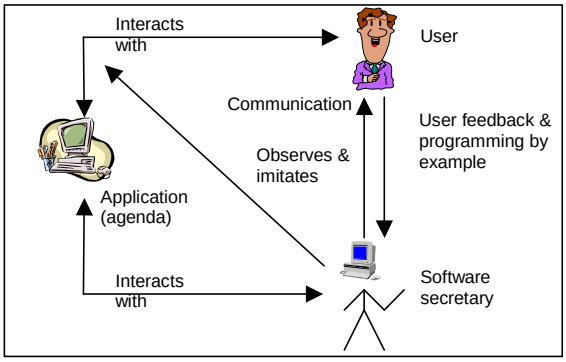

Figure1. How an agenda agent works

2.1 Interactions among a user and its agenda agent

There are several ways in which an agent can interact with its user: the agent can notify or alert the user about situations relevant to him; the agent can suggest the user some meeting parameters or the execution of actions; the user can ask the agent to perform some task on his behalf or he can ask it for advice; the agent can request some information from the user or can ask him to make a decision; the agent can execute actions autonomously, notifying the user afterwards. In the following paragraphs we will provide some detailed examples of each type of interaction.

Alerting the user The agent can alert the user about particular situations that may be relevant to him. Some of them are listed below:

- The agent can notify (it has to!) the user about the actions it has autonomously performed on his behalf. For example, the agent notifies the user that it has sent an email to his boss saying he will attend a meeting the following day and that it has scheduled that event.
- The agent can alert the user about conflicting events when the user is scheduling a new event.
- The agent can notify the user about incoming events some time before the event will take place. For example, the agent can tell the user he has a business meeting with Project A's team in half an hour.
- The agent can alert the user about particular events related to his contacts such as birthdays and anniversaries.

Asking the agent for help and to execute tasks The user can ask the agent for help when he is scheduling events and he can also ask the agent to execute some tasks. For example:

- The user is scheduling a new event and he asks the agent to suggest him some event parameters given some information. For example, suggest the event date, event duration, event time and event place given the event type and the participants.
- The user has two conflicting events and wants to reschedule one of them. So, he asks the agent to suggest him alternative dates and times for the event.
- The user is organizing an event in which many people is involved and he asks the agent to determine appropriate meeting dates, meeting times and meeting places for everyone. The agent has to manage, in this case, the communication and negotiation with the other participants.

Making suggestions to the user The agent can make suggestions to the user when he detects some particular user actions or some particular situations:

- The agent detects that the user is scheduling a new event and it decides to suggest him some event parameters according to the information the user has already entered. For example, the agent can suggest a meeting time, a meeting date and a meeting duration according to the meeting participants and meeting topic.
- The agent detects that the user is scheduling a new event that originates a conflict with an already scheduled one. Thus, it suggests the user another meeting time and date because the scheduled event is more important.
- The user receives an email in which a contact invites him to attend an event. Thus, the agent suggests an answer to the user plus a justification of its suggestion. For example, it can advice the user to accept the invitation because the host is his boss.
- The user is organizing a meeting and he receives mails in response to his invitation. Thus, the agent suggests him what to do in the case of a rejection or a counterproposal.

Asking the user to take decisions When the agent is autonomously executing tasks, it can ask the user for information or for confirmation:

- The agent is rejecting an invitation because the user has another scheduled event, but it first asks the user for confirmation.
- The agent is scheduling an event for the user and it asks the user to choose between two possible days for the event.
- The agent is going to send notifications of an incoming event organized by the user to a list of participants, but it first asks the user for confirmation.

There are also more complex interactions between a user and its agenda agent, which involve a conversation between them, as shown in Figure 2. Consider

the following example: the user is organizing a meeting with some employees and he decides to ask the agent for help. Thus, he asks the agent to suggest him some possible meeting dates and meeting times. He also gives the agent the information of the event: the topic of the meeting, the estimated duration and the participants. Then, the agent suggests the user a set of possible dates and times, ordered according to the user's priorities. The user can then execute different actions, such as choosing one of the suggestions and scheduling the event, rejecting all the suggestions and determining the date and time by himself, accepting partially one of the suggestions (e.g. the date) and asking the agent to make another suggestion. In this last situation, the conversation will continue until the user ends it.

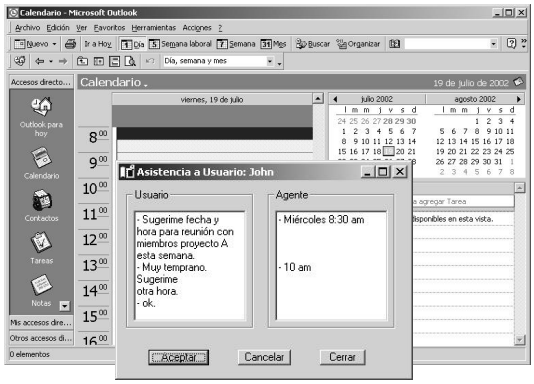

Figure2. Typical conversation between user and agent

3 Architecture of a software secretary

In the previous section we have enumerated the different interactions between a user and its software secretary. In order to achieve these interactions the software secretary must have various capabilities: observing the user's behavior while he is working with the agenda system, learning the user's scheduling preferences and habits from this observation, learning about the user's contacts and their scheduling preferences, deciding when and how to assist its user and executing actions autonomously to provide assistance to its user. These capabilities distinguish our software secretary from the current available computer programs that manage calendars. Figure 3 shows the different architectural components of our agenda agent, which provide it the capabilities it needs to behave as a secretary.

3.1 Observation

In order to acquire information about its user, our agenda agent observes a user's behavior while he is working with the agenda system. This is achieved through

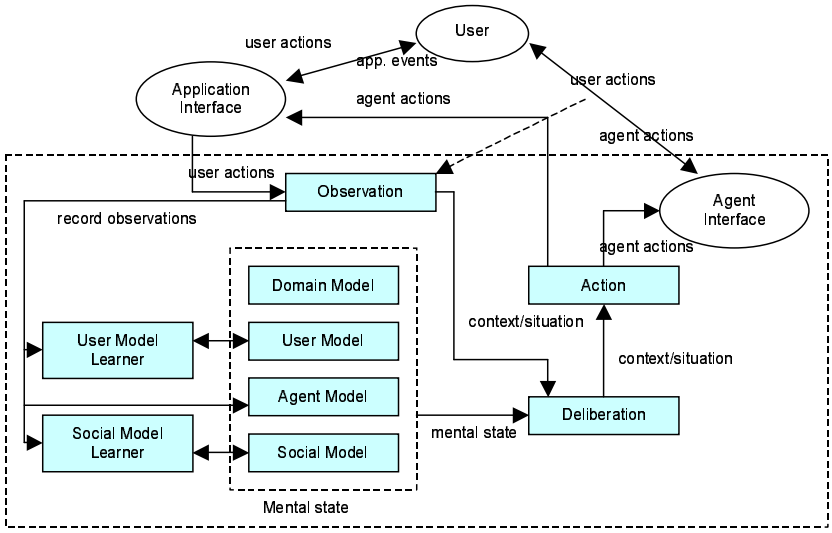

Figure3. Agenda agent's architecture: general view

the *Observation* component, as shown in Figure 3. This component captures each action that the user executes when he interacts with the agenda system (through the *application interface*) and with the agent (through the *agent interface*). The kinds of actions the user executes, when he executes them and what parameters he uses give an agent an idea of the user's scheduling priorities and preferences. For example, when the user is scheduling a meeting the meeting date, the meeting time, the duration and the meeting type tell the agent about the user's scheduling preferences. The cancellation of a previously scheduled event because of a new event, tells the agent about the user's priorities. The user actions that an agenda agent should record are the ones listed below:

- The user schedules a new event: the agent has to observe and record the parameters of the scheduled event in order to determine the characteristics of the events the user schedules and the relationships among them. The agent has to record the event parameters discussed in Section 2.
- The user modifies a scheduled event: the modifications the user performs tell the agent about the user's scheduling priorities and the way in which the user relaxes these priorities.
- The user cancels a scheduled event and places a new event into the same time slot: this situation tells the agent that the new event is more important for the user.
- The user sends an email inviting some of his contacts to an event he is organizing: this action provides the agent information about a user's contacts, about the relationships between the user and his contacts and about the events that relate a user with his contacts.

- The user answers an invitation he has received. This answer may be an acceptance, a rejection or a counterproposal to the invitation. This situation tells the agent about the scheduling preferences and priorities of its user and his contacts.
- The user answers a counterproposal for an event he is organizing. He can accept the counterproposal, reject it or make another one. This action tells the agent about the user's preferences and the way he relaxes them in order to reach an agreement.

The *Observation* component records the information obtained from observation into the *Mental State* of the agent. This information is processed by the learning component to build the user model or user profile.

3.2 Learning and knowledge management

As we have already mentioned, one of the main capabilities a software secretary must have is learning the user's scheduling preferences and habits. An agenda agent learns about a user through three different sources: through the examples explicitly provided by the user, by observing the user's behavior and through the user feedback. The most direct source of learning are the priorities and scheduling preferences that the user (optionally) provides. This information is very useful to assist the user since it represents concrete user preferences. However, most users do not like to spend their time entering data and they probably do not know how to express their preferences. Thus, the agent will not always count with this source of learning. Some of the preferences a user can provide are the following:

- The user can provide his priorities for the different event parameters. For example, for some user the meeting time may have a high priority while the meeting place may have a low priority.
- The user can provide his preferences for the different values each event parameter can adopt. For example, if we consider the meeting times, the user can tell the agent that he hates scheduling business meeting before 10 a.m. and after 7 p.m.
- The user can inform the agent about regular events such as gym classes, courses he teaches or weekly dinners with his friends, which have a high priority and cannot be rescheduled.
- The user can provide priorities combining different event parameters. For example, the meeting time usually has a high priority for the user, but if the meeting host is the user's boss this priority becomes lower.

An agenda agent also learns from the feedback the user gives regarding the assistance the agent has provided him. This feedback may be explicit or implicit. It is explicit if the user gives his opinion or rates the agent's assistance through a user interface provided for this purpose. However, if the user does not explicitly provide it, the agent has to detect it by observing the user's actions. The following actions reveal positive and negative feedback regarding the agent's tasks:

- The user executes an action suggested by the agent, such as rejecting an invitation.
- The user schedules an event using the parameters suggested by the agent.
- The user does not execute an action suggested by the agent and performs another task.
- The user schedules an event without considering the parameters suggested by the agent, using others instead.

The *Observation* component records the information obtained from observation and the information explicitly provided by the user into the *Mental State* of the agent. The mental state of the agent contains all the information the agent uses to take decisions and to execute actions. It contains information about the application domain (*Domain Model*), about the user (*User Model*), about the user's contacts (*Social Model*) and about itself (*Agent Model*).

The *Domain Model* contains information about a particular application domain. In our example, it can contain information about meeting places or a city map. The data stored in this component may be provided by the user, by the designer or by other applications. The representation of this information may adopt different forms: a database (of meeting places, for example), a set of rules, among others.

The *User Model* contains both raw data about a user's actions recorded by the *Observation* component and information about a user's priorities, preferences and behavioral patterns inferred from the raw data. The inferred information constitutes the user profile. The user profile is composed, in this domain, of those things that an agenda agent must know about its user in order to assist him: what tasks the user performs when he uses the agenda system, the parameters or characteristics of these tasks, the priority each action and action parameter have for the user, when the user executes each type of action, the relationships between certain parameters and the relationships among different actions.

In order to build a user profile, the software secretary must have the capability of processing the information obtained by the *Observation* component and deriving the scheduling preferences, priorities and habits of its user. This task is achieved by the *User Model Learner* component. The *User Model Learner* uses one or various techniques to derive a user profile from his actions, his explicit preferences and his feedback. For example, in its simplest form, it can analyze the correlation among the different meeting parameters in order to determine relationships among them. The same technique may be used to determine relationships among different user actions. The user's preferences and priorities may be determined through probability calculus (frequencies). Other techniques such as Case-Based Reasoning [11] and Bayesian Networks[8] can be also applied. Figure 4 shows a Bayesian Network modeling some intuitive relationships among event parameters for a given user.

If an agenda agent wants to assist its user even better it also has to acquire knowledge about the relationships the user has with other people and the activities they share. In this way, the agent can, for example, arrange meetings with a user's clients taking into account not only the user's priorities but also

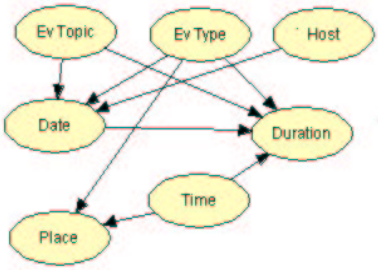

Figure4. Bayesian Network modeling event parameters' dependencies

his clients' priorities. The *Observation* component also captures those user and application actions involving other participants: invitations other people send to the user, invitations the user sends to his contacts, replies to the user's proposals (acceptances, counter-proposals, rejections), the user's replies to his contacts' proposals (acceptances, counter-proposals, rejections). This component records the information about actions involving a user's contacts into the *Mental State* of the agent, particularly into the *Social Model*. *The Social Model Learner* component takes this information and using some Artificial Intelligence techniques it builds the social profile of the user. The social profile contains information about a user's contacts, the relationship the user has with each contact (work-mate, friend, boss, relative), the importance each contact has for the user, the kinds of events the user and each contact attend together and the scheduling preferences of each contact, both when he is a host and when he is an invitee.

3.3 Personality

One of the components of the agent's *Mental State* is the *Agent Model*, which represents the personality of the agent. The "personality" of an agent is a very important aspect in agent development since it reflects the way in which the agent behaves and the way in which the agent interacts with its user. An agent may be trustful or not; it may interrupt the user constantly to tell him something or just in very important occasions; it may be pro-active or not, asking always the user for permission; it may be discrete or it can tell everyone about the user's scheduling priorities. A combination of these aspects will lead to a given agent personality. Since each user interacting with an agenda agent will probably like a different type of agent, being able to choose the agent personality is an important issue.

The personality of an agent is defined by two parameters: the confidence level and the interaction level. The confidence level represents how trustful the agent is. The agent determines this by evaluating the agent's behavior. If the user asks the agent for advice and it makes good suggestions the confidence level will increase. However, if the agent makes mistakes when helping the user, the confidence level will decrease. Closely related to this confidence level is the "freedom" the user gives to the agent. Then, if the confidence level is high the

user will let the agent execute actions on his behalf, but if the confidence level is low he will only allow the agent to make him suggestions. The confidence level is indicated by a set of thresholds: when this level is greater than a given threshold value the agent can perform certain actions. For example, in a 0 to 100 scale, when the level is below 30 the agent can notify the user about relevant situations such as incoming events; if the confidence level is between 30 and 60 the agent can also answer to a user's requests; if the level is between 60 and 90 the agent can also make suggestions to the user; and with a confidence level higher than 90 the agent can execute some actions autonomously. Confidence levels may be associated to every agent action or they can be associated to particular agent actions.

The interaction level indicates the agent when and how often to interrupt the user. Some users do not like to be interrupted when they are working, but some others do not mind being disturbed. The user can select how much his agent will interact with him: only when the user requires it (he asks the agent for advice or asks the agent to do something); to notify him about certain situations (the user can choose these situations from a list); to make suggestions with a given certainty level (the user can select the types of suggestions); to ask the user for information (always, in certain occasions, when it is going to execute an action, never). The user can select any combination of these.

3.4 Decision

In order to assist its user in a proper way, the agenda agent must have the capability of deciding when to assist him and how to assist him. The *Observation* component notifies the *Deliberation* component about those user actions that require some agent action. For example, upon reception of a user request the *Deliberation* component has to decide what to answer him; when the user is executing an action in which the agent can provide help, this component has to decide what to suggest; when receiving a reply to an agent's question this component has to decide how to manage this answer. Once the agent has made his decision, the *Action* component executes the correspondent agent actions, both through the *agent interface* and through the *application interface*.

The agenda agent has to determine:

- situations or contexts in which the agent can provide help or assistance to the user in a pro-active way. For example, sending a notification alerting the user about a meeting the user has in half an hour and suggesting meeting parameters when the user is scheduling a new event.
- situations or contexts in which the agent can execute actions autonomously on behalf of the user. For example, refusing an invitation because it does not meet the user's preferences.
- situations or contexts in which it must initiate a dialog with the user. For example, asking for confirmation before scheduling a meeting or asking the user to choose between two possible meeting dates.

Besides, the agent has to decide what to do in each of these situations, i.e. what to suggest, what action it should execute, what to ask, what to reply to a user's request, how to carry on a conversation with the user. To achieve these goals, the agent uses its "brain" represented by the *Deliberation* component. This component is in charge of deciding what the agent has to do in a given situation. In order to take decisions the agent considers a particular situation occurring at a given moment and the context in which it happens, and then it decides what to do. The *Deliberation* component uses the data contained in the agent's mental state to guide the decision making process. Several techniques may be used to implement this component such as Influence Diagrams [10], Rule Systems and Decision Trees.

3.5 Action

The agenda agent executes different actions to assist its user through the *Action* component. Some of these actions involve interactions with the user (through the *agent interface*) and others involve interactions with the application (through the *application interface*). In Section 2.1 we have explained in detail those actions in which the agent interacts with its user to notify him about situations relevant to him, to answer a user's assistance request, to make him suggestions or to ask him for information or to take decisions. The agent can also execute some tasks autonomously without interacting with its user, such as sending mails to the participants of an event organized by the user to remind them about the event and refusing an invitation for a party the agent is sure the user will not attend because he has scheduled a business dinner at that time. The types of actions the agent can execute depend on the agent personality, as we have explained in Section 3.3.

4 Conclusions and future work

In this work we have shared our experience in the development of interface agents. We have presented an architecture that enables interface agent developers to build software secretaries that can assist users in a personalized way, as human secretaries do. We have described how the different architectural components of our proposed architecture and the relationships among them have been materialized into an agenda agent. This is just one of the domains in which our architecture can be applied. We are now working on other application domains, and the results obtained so far indicate that the design solution prescribed by our architecture will lead to efficient software secretaries.

Since one of the knowledge sources of interface agents are other agents, we are also extending our proposed architecture so that software secretaries will be able to communicate and negotiate with others to assist their users.

References

1. http://www.russellinfo.com/cmdesc.html.

2. http://www.milum.com/html/otptour1.html.
3. http://www.abs-usa.com/products/vgp/.
4. Jeffrey Bradshaw. KAoS: An open agent architecture supporting reuse, interoperability and extensibility, 1996.
5. Hans Chalupsky, Yolanda Gil, Craig Knoblock, Kristina Lerman, Jean Oh, David Pynadath, Thomas Russ, and Milind Tambe. Electric elves: Applying agent technology to support human organizations. In *International Conference on Innovative Applications of AI (IAAI'01)*, 2001.
6. D. Cockburn and Nicholas Jennings. ARCHON: A distributed artificial intelligence system for industrial applications. *Foundations of Distributed Artificial Intelligence*, pages 319–344, 1996.
7. Leonardo Garrido and Katia Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *International Conference on Multi-Agent Systems*, pages 95–102, 1996.
8. D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-9506, Advanced Technology Division - Microsoft Research, 1995.
9. N. R. Jennings and A. J. Jackson. Agent based meeting scheduling: A design and implementation. *IEEE Electronics Letters*, 31(5):350–352, 1995.
10. Finn V. Jensen. *An Introduction to Bayesian Networks*. UCL Press, 1996.
11. J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, 1993.
12. Robyn Kozierok and Pattie Maes. A learning interface agent for scheduling meetings. In *Intelligent User Interfaces 93*, pages 81–88, 1993.
13. Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, 1994.
14. Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):81–91, 1994.
15. J. Muller and M. Pischel. The agent architecture InteRRap: Concept and application. Technical Report RR-93-26, DFKI Saarbrucken, 1993.
16. Terry Payne, Rahul Singh, and Katia Sycara. Browsing schedules: An agent-based approach to navigating the semantic web. *First International Semantic Web Conference*, 2002. (to appear).
17. Nwana H. S. Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.
18. Sandip Sen. Developing an automated distributed meeting scheduler. *IEEE Expert*, 12(4):41–45, July/August 1997.
19. Sandip Sen and Edmund Durfee. On the design of an adaptive meeting scheduler. In *Tenth IEEE Conference on AI Applications*, pages 40–46, 1994.
20. Sandip Sen and Edmund Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7:265–289, 1998.