Open Sesame!: A Look at Personal Assistants*

Michelle A. Hoyle and **Christopher Lueg**
AI-Lab, Department of Computer Science
University of Zurich
Winterthurerstrasse 190, CH-8057 Zurich
Tel. +41-1-257 4577 Fax +41-1-363 0035
{hoyle,lueg}@ifi.unizh.ch

Abstract

Software agents, or softbots, are supposedly intelligent programs that assist the user in performing repetitive, boring, and time-consuming tasks. An overview of software agents, with special attention focused on the so-called personal assistants, is presented as an introduction. Following that, an in-depth look at one of the first commercially available intelligent agents Open Sesame!, a personal assistant for managing the Macintosh desktop environment, is presented. The existing intelligent agent paradigms, we argue, break down when it comes to actively helping the end user because of flaws inherent in the current approaches to intelligent agent design. To that end, we discuss the reasons why this is, particularly with respect to Open Sesame!, and promote the notion of situatedness to be taken into account more seriously for developing software agents that actively support the user for the duration of the task at hand. As an example for a more situated approach to personal assistant design, a quick overview of a project to develop a personal assistant for news filtering is discussed.

Introduction

In the following we report on several months experience with one of the first commercially available software agents Open Sesame!, a personal assistant for the Macintosh computer. We start initially with an overview on software agents, focusing on personal assistants. Following that, we briefly introduce Open Sesame! and present observations made during a nine month period of daily use of the agent. We discuss these observations with respect to user modeling and situated action, and propose an alternative approach to agent design which is inspired by two years of work on Situated Design, a methodology for designing computer systems being developed in our lab over the past few years. To illustrate our points we describe a situated approach to Usenet News filtering, where the user is actively supported during the filtering task. Finally, we conclude by reiterating the lessons the last decade of intelligent agent design has taught us and look toward a more realistic future involving agents that work hand-in-hand with their human counterparts.

Software Agents

Currently the limelight of public and scientific interest is focused on software agents, also called softbots, with many new commercial products being released that supposedly make use of artificial intelligence techniques. Even entire issues of popular science magazines have been devoted to the topic [c.f. Scientific American, September 1995]. Unfortunately, with all this publicity, software agents are being hyped at a dangerously high level due to overblown and partly unrealistic expectations (Norman 1994; Jennings & Wooldridge 1996). Therefore, it is necessary to carefully inspect that we can realistically expect from software agents in terms of performance and responsibilities. Within the agent community itself, there is no yet commonly agreed upon definition of what exactly constitutes an agent. One central point, however, seems to be autonomy, or the agent's ability to control its own behavior to a certain degree (Foner 1993; Jennings & Wooldridge 1996; Etzioni, Lesh, & Segal 1994). Other points mentioned in the literature, e.g., (Jennings & Wooldridge 1996), are so-called social abilities such as the ability to exchange data with other agents, responsiveness to the environment, and proactiveness.

Nevertheless, it is still unclear what distinguishes software agents from other computational entities, e.g., processes in operating systems or objects in object-oriented languages. Some researchers view agents as being conceptually more abstract entities than objects (e.g. (Shoham 1993)). Others classify computational entities as agents depending on the tasks they are to

* Proceedings of the International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 97) London, 21.–23.4.97, pp. 51–60.

complete; for example, interface agents that aid a user or personal assistants. The classification of entities as agents according to their computational complexity, i.e., as gopher agents, service performing agents, and "predictive" agents (Jennings & Wooldridge 1996), is also commonly used.

Application areas for software agents range from distributed artificial intelligence (DAI) (Rao & Georgeff 1995; Wooldridge & Jennings 1995a) to interactive simulation environments (Tambe *et al.* 1995) and entertainment (Mock, Lawton, & Hoyle 1996). In the context of increasing interconnectivity, mobile computing is of increasing interest (Harrison 1995). Mobile agents are capable of roaming entire networks and traveling from one machine to another (Coen 1994; Gray 1995; Gray, Rus, & Kotz 1996; General Magic, Inc. 1995; Sun Microsystems, Inc. 1996). Other recent approaches to software agents are strongly influenced by Artificial Life, where biological tenets of evolution and genetics are applied to the development of agents. The application of artificial life techniques to an information filtering problem produced Amalthea (Moukas 1996), an agent capable of filtering through online WWW documents. Comprehensive overviews of software agents in general are available from Wooldridge and Jennings (Wooldridge & Jennings 1995b) and Nwana (Nwana 1996).

Since agents are being unrealistically hyped, the challenge is to select a task that is both realistically solvable and of significance to real users. Therefore, this paper focuses on personal assistants, both in fact and in theory.

Personal Assistants

With the advent of graphical user interfaces (GUI), direct manipulation has become the dominating metaphor in human-machine communication, i.e., the user must initiate all tasks explicitly and monitor all events (Maes 1994a). In recent years, the idea of personal assistants supporting humans in their work has emerged. The primary thrust of this trend is that computer programs take over boring, or repetitive and time-consuming tasks in order to increase human productivity and creativity.

The main application areas so far are in helping humans to cope with supposed information overload and to assist users in performing repetitive, common tasks. Information filtering and information seeking are two approaches to deal with information overload caused by modern communication media like Usenet, email, or the World Wide Web (WWW). Information filtering tries to reduce the amount of incoming information to a comfortable amount by the application of information retrieval methods. An example of such an application is Letizia, a WWW filtering agent developed at the MIT Media Lab by Henry Lieberman (Lieberman 1995). Letizia attempts to infer from documents the user has examined what other currently accessible documents might be of current interest to the user, where current is defined by some degradation function. She can then make recommendations, which the user may or may not heed which provides a type of feedback as to the helpfulness of the suggestion.

Information seeking aids the user in finding interesting information, which grows increasingly difficult for a user because of the large amount of material now available electronically. The goal here, then, is to search for interesting information automatically to relieve the user of tedious search tasks, e.g., (Balabanović & Shoham 1995; Armstrong *et al.* 1995; Lieberman 1995) and, of course, WWW search engines[1].

Email handling and meeting scheduling are examples of applications to help users in performing repetitive tasks. The former allows for the automatic forwarding, storage, and deletion of electronic mails, freeing the user from the necessity of coping with electronic junk mail (Denning 1982), or performing repetitive tasks such as storing mails from a mailing list or deleting unwanted requests. A personal assistant for meeting scheduling incorporates an agent who can negotiate a meeting in line with the preferences of its owners. Such an assistant was developed at AT&T Bell Labs by Henry Kautz and Bart Selman (Kautz *et al.* 1994; Kautz, Selman, & Coen 1994). KautzBot and SelmanBot, the names for their scheduling agents, communicated with other agents via email and attempted to automatically negotiate an acceptable meeting time for all involved participants. Methods for both these applications include user-defined models of interest, e.g., rule-based kill files or email filters, and automatically acquired user models.

Other approaches to assisting users belong to the class of human-computer interaction (HCI). Here the idea is to monitor user activities and to detect repetitive sequences. Examples range from simple purely event-driven approaches like macro recorders to more complex approaches doing some processing to uncover repetitive tasks like Eager (Cypher 1991). Most people are already familiar with macro recorders which appear in many text editors and word processors. Applications such as Eager analyze a collection of accumulated events and attempt to deduce a pattern that might be automated out of the collection. Both these types of applications tend to be quite limited in what they can accomplish as the algorithms driving them

[1] See (Koster 1996) for a comprehensive overview

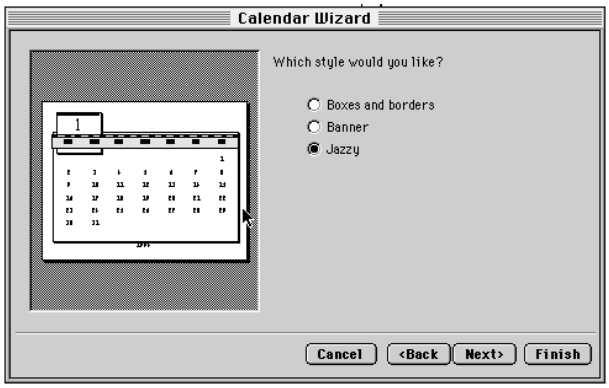

Figure 1: Microsoft Word's calendar "wizard"

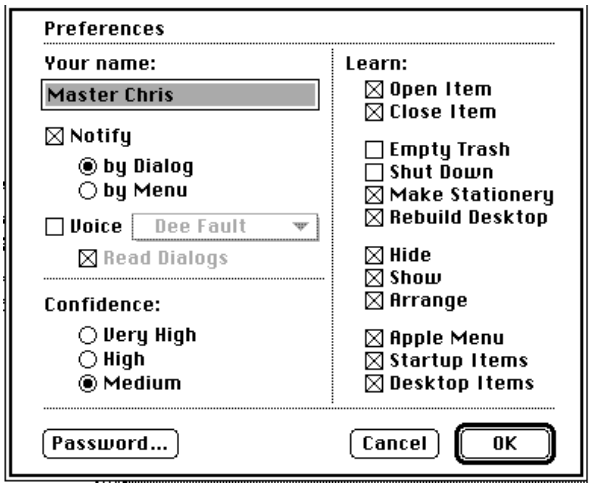

Figure 2: Open Sesame's preferences

are very literal minded, not permitting much deviation from what was recorded originally, making use of explicit keywords and so forth.

Most of the above approaches are still within the realms of academia; however, some beta-releases of academic products are available to the public. For example, the MIT Media Lab's email filtering agent Maxims, which allows the agent to contact other email agents for clues about how to handle pieces of email if there are no precedence rules for the current piece of email (Lashkari, Metral, & Maes 1994; Maes 1994b), is available as a beta-release. Commercial products have made use of some more complex approaches to provide so-called assistants, or wizards, to monitor user activities in modern word processors and spreadsheets and propose enhancements or faster ways of accomplishing tasks in the case of ponderous behaviour. For example, Microsoft Word provides "wizards" (See Figure 1). A wizard is just a template that helps the user create particular types of documents quickly by answering some questions. In order to create a calendar document, the user is asked if they want boxes around the calendar, which of three title banners they want, etc. One of the first commercially available "intelligent software agents" (Charles River Analytics 1996) was Open Sesame! for the Macintosh, which we describe in the next section and report on several months of experience with the agent.

Open Sesame! – A Commercially Available Software Agent

The commercially available software package Open Sesame! from Charles River Analytics was announced as being the first learning assistant for the Apple Macintosh and Microsoft Windows (Charles River Analytics 1996). Its task is to "eliminate mundane, time-consuming tasks so that every minute you spend at your computer is productive." Open Sesame! observes user activity and learns which tasks the user performs repeatedly. It then performs these tasks automatically. Charles River Analytics expects intelligent agents to "radically change the way we use computers, allowing software to be an assistant to the user." They even expect that "the impact of software agents on personal computing can be equal to, if not greater than, the impact that graphical user interfaces on personal computing when they replaced command line user interfaces."

The 1.1 version of Open Sesame! tested requires an Apple Macintosh with MacOS 7.0 or higher. Tests were performed using a Macintosh Quadra 700 with MacOS 7.01 and 8 megabytes of main memory and Power Macintosh 8100 with MacOS 7.5.3 and 16 megabytes of memory. Both computers were used frequently for office work, email and Usenet news handling, and web browsing. The system configurations also included, among other things, commercial packages like a word processor, a spreadsheet, a newsreader, an email program, and a web browser. Both computers are connected to a LAN via Ethernet, and from there to the Internet.

Open Sesame! observes user activity by monitoring Apple events[2] and inferring repetitive patterns. It sends Apple events directly to the Finder and scriptable applications without using the AppleScript[3] engine. Open Sesame! distinguishes between time-based

[2]Apple events are high-level process instructions that can be sent to almost any process running on a Macintosh under a MacOS 7.x or greater environment. At a minimum, most applications support the ability to start, quit, and open and close a file through Apple events.

[3]AppleScript is Apple's language for the exchange of data between applications through Apple events.

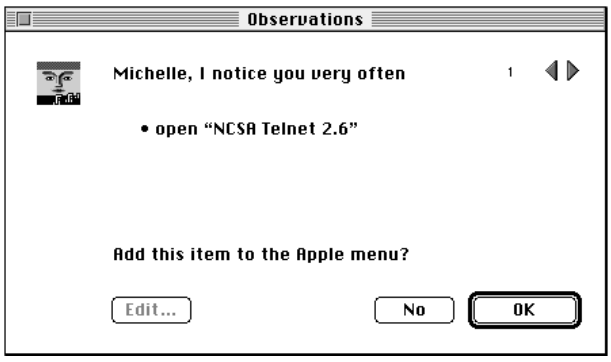

Figure 3: A suggestion by Open Sesame!

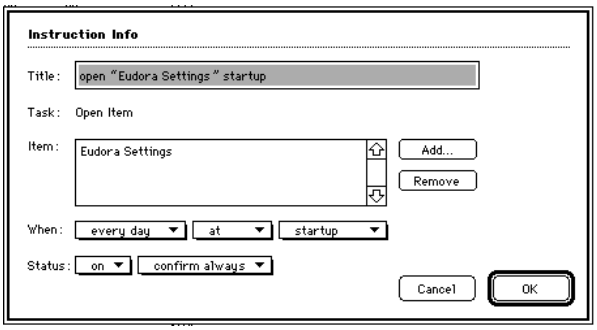

Figure 4: Manipulation possibilities offered by Open Sesame!

and event-based triggers. A time-based trigger means that Open Sesame! observes a certain user action repeatedly at the same time, e.g., every Monday at 9:00 am or every full hour. An exact match is not required. 9:00 am also covers some minutes before or after 9:00 am. An event-based trigger means that Open Sesame! observes a sequence of certain desktop actions repeatedly, e.g., opening a folder after closing Eudora. Figure 2 shows some preferences offered by Open Sesame!. The user can choose a certain degree of anthropomorphization (e.g., notification by a female voice), a confidence threshold, and user actions to be considered for learning.

The manual for Open Sesame! mentions that some neural learning mechanism is used but does not give further explanations as to what kind of neural network or which learning rate is used. (Caglayan *et al.* 1996), however, claim that Open Sesame! makes use of a variation of the adaptive resonancy theory-2 (ART-2) algorithm of Carpenter and Grossberg (Carpenter & Grossberg 1987). The implementation of the ART-2 algorithm allows Open Sesame! to monitor and to recognize patterns relating to the opening and closing of documents or folders; running and quitting of applications; user preferences with respect to hiding, showing, and arranging various objects in the desktop environment; and user preferences with respect to desktop management, startup items, and objects occurring in the Apple Menu "...through competitive filtering of patterns for finding the 'best match' category, combined with top-down template matching for determining if the best match is 'good enough."'

The user is able to edit suggestions (see figure 3) made by Open Sesame!. The program always asks whether the user welcomes the actual suggestion at all. If the user indicates that the suggestion is unacceptable, the program promises never to make that observation again. If the user welcomes the suggestion, then the user is invited to modify the instruction suggested by Open Sesame! prior to the suggestion being incorporated into Open Sesame!. Figure 4 shows the facilities to tune the instruction, e.g., the status of the instruction at all, the time of the suggested action, and the requirement of confirmations prior to execution of the instruction. A tuning facility is provided in a rule-based manner. Unfortunately, the manual does not mention how these manually modified tunings are considered by Open Sesame!

Open Sesame! was tested on the Apple Macintosh Quadra 700 and a Power Macintosh 8100 for a period of more than 9 months. During this period, Open Sesame! made 129 different suggestions (see figure 5 for some examples). 70 suggestions were related to programs installed on the computer (31 time-based, 39 event-based) and 59 were directly related to desktop or Apple menu settings (all event-based). The subjects accepted only 2 out of 129 suggestions immediately. Several others were not immediately accepted but considered for later use.

Both positive examples related to programs that were used frequently: Eudora, an email handling program, and NewsWatcher, a Usenet newsreader, respectively. Eudora is usually opened right after starting up the computer. Newswatcher is started irregularly for reading Usenet news. In the case of NewsWatcher, the subject did not accept an early suggestion to launch the program right after startup because its use depends on having the time needed to read news. A later accepted suggestion applied to NewsWatcher's subscribed newsgroups file. However, it was necessary to adapt both suggestions manually due to a special installation of both programs being used by several users.

All suggestions not immediately accepted but considered for later use were related to desktop and Apple menu settings. Open Sesame! correctly observed the user having repeatedly used particular folders, e.g., the folder "COOP96" being used for the storage of various paper revisions during the writing of that paper.

However, the process of writing the paper being only a temporary one, the subject did not agree to add an alias for this folder to the desktop. Another example was the proposal of a new desktop icon representing the "AILab" folder. Among other things, the lecture notes for the current course on real world computing are to be found in this folder. Since we are rewriting parts of the course notes during this term, the observation made is correct, of course, but this and other observations are not important enough to warrant cluttering up the desktop with icons. An example for a suggested removal of an item from the Apple menu was "WebWeaver." Admittedly, the program has not been used for a quite long time but, nevertheless, the subject did not agree to remove the icon for a reason that is not very obvious: The program's presence continuously reminds the subject of having intended to update his WWW homepage a long time ago.

Aside from the positive examples discussed so far, there were no other suggestions by Open Sesame! that were even considered. Some suggestions were event-based and reflected repetitive patterns in the usage of some programs, like the opening of BBEdit Lite right after Simpletext. The reasons for doing this are manifold: Simpletext, for example, is unable to handle large textfiles (this could be taken into consideration based on the actual size of the text) or BBEdit has more sophisticated editing facilities. Obviously, the correct choice of an editor depends heavily on the editing context. Probably a better suggestion would have been to simply replace Simpletext by BBEdit. Open Sesame! also observed many time-based events such as opening NCSA Telnet on Friday at 14:30h or opening Fetch, a file transfer utility, at 14:15h. There is simply no plausible reason why these programs should be opened at that particular time; their use depends on need, not time of day.

Based on customer feedback, some of the points mentioned above have been addressed by the developer. Improvements slated for the 2.0 version of the product include less dependence on time-based learning and more on event-based learning, additional monitoring capabilities of third-party applications, incremental processing of data, in-context coaching to teach users how to use features they are unaware of, five times more types of events, and the ability to specifiy conditions on instructions (Caglayan *et al.* 1996).

Monitoring User Behavior and Situated Action

Based on our experiences with Open Sesame!, we conclude that Open Sesame! did not provide what we expected from an intelligent personal assistant, although we must admit that many of the capabilities mentioned by the developer were never seen, such as creation of document templates for document formats that are reused, etc. Of course, satisfaction of expectations from a product is inherently subjective; others may be quite satisfied with the performance of Open Sesame! This ambivalence is clearly shown in professional reviews. For example, the magazine *PC AI* (Rasmus 1995) shows enthusiasm about the product claiming that "Open Sesame! [...] is perhaps the most innovative of the agents currently shipping" while *MacUser* (Markoff 1994) found Open Sesame's suggestions "dim" but at least useful for people that "use a Mac in a fairly routine manner – turning it on each morning, opening a few favorite programs, and then reading your e-mail."

Open Sesame!'s failure as a personal assistant–as we experienced it from an intelligent agent–can be traced down to two distinct explanations: on the one hand, there are technical reasons why Open Sesame!'s abilities are limited in the actual domain, but on the other hand, theoretical considerations indicate general limitations for (electronic) personal assistants.

Technical problems range from a lack of support from the operating system in monitoring user behavior to problems with the amount of observed patterns of behavior. The former, lack of operating system support, led the developers of Open Sesame! themselves, Charles River Analytics, to halt the development of Open Sesame! for the current Macintosh operating system, claiming that "the MacOS does not make it easy to monitor high level user behavior reliably by a third-party background application" (Caglayan 1996).

An insufficient amount of *useful* behavior patterns may arise for different reasons: For example, there may just not be enough patterns at all to be observed by an assistant or the learning algorithm is not good enough to detect the patterns. This seems not to be the case since Open Sesame! presented far more than hundred observations while being tested and Charles River Analytics actually decided to extract the learning algorithm and use it for a learning engine sold seperately from Open Sesame!. This may be interpreted as degree of confidence in the performance of the underlying algorithm.

However, it seems not only be technical problems that cause the Open Sesame! to behave as "nagware" (Markoff 1994) instead of being a personal assistant as expected. Rather, it is the notion of situatedness (Suchman 1987) being fundamentally neglected in personal assistant design. In the following section we briefly introduce the notion of situatedness and we point out some implications for the design of personal

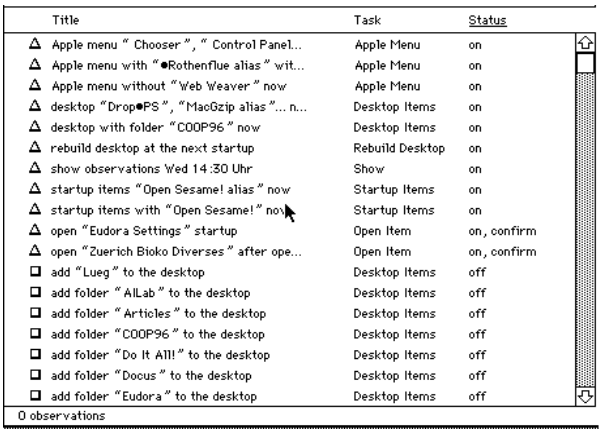

Figure 5: Some suggestions made by Open Sesame!

assistants. Then we briefly introduce an actual project using an alternative approach to personal assistant design in order to illustrate our point. The example is taken from a different domain, information filtering, but both approaches to personal assistants are based on monitoring user behavior. However, they differ in the implications drawn from considering situatedness.

Situatedness and Personal Assistants

Some of Open Sesame!'s non-technical problems (i.e., problems apart from those caused by the lack of operating system support, etc.) may be interpreted as being the same problems as experienced in expert system design. The most prominent problems in this domain are the frame problem (Pylyshyn 1988), the frame of reference problem (Clancey 1991), the symbol grounding problem (Harnad 1990), and the lack of situatedness (Suchman 1987). Since a basic discussion of all these problems is far beyond the scope of this paper,[4] we focus on the lack of situatedness since it is the most important problem in this domain and is directly related to the observations made during the experiences with Open Sesame!. Situatedness may be used to explain why Open Sesame!'s generalization capabilites do not always reflect what users expect.

An example from daily work given by one of the professional reviewers of Open Sesame! (Markoff 1994) beautifully illustrates the implications of lacking situatedness. Open Sesame! correctly observed that the user has emptied the trashcan several times right after dragging a document into it. However, according to the reviewer, it is not correct to deduce that the trashcan should always be emptied when a document is dragged into it. Emptying the trashcan did not depend only on the *action* of dragging the document into the trash but also on the *content and type* of the document, i.e., the document containing confidential or private data. It is important to note that judging the relevance of a document in a situation is a problem in principle and not a particular deficiency of Open Sesame!: the importance is dependent upon the situation the user is involved, the content of the document, the relation between the author and the reader, etc. Accordingly, the significance of the document may not be judged at all by an unsituated personal assistant (at least not by an electronic one).

Viewing humans as situated actors in contrast with computers explains why Open Sesame!'s generalization capabilities are sometimes correct (in a mathematical sense) but nevertheless inappropriate. Both humans and computers performing real-world tasks necessarily interact with the real world and not with electronic representations of the world. In order to avoid overblown and partly unrealistic expectations of software agents, it is necessary to consider the fundamental differences between the way humans and computers interact with the real world. As a matter of fact, the real world is constantly changing, intrinsically unpredictable, and infinitely rich. Therefore, it follows that the real world is always subject to limited perceptions and is always only partially knowable (Pfeifer & Rademakers 1991). Humans have evolved to cope with this intrinsic vagueness. They behave as situated actors (Suchman 1987), meaning that they are able to use the world as its own best model and to bring to bear their experience onto the current situation. In an extreme contrast, computer programs, e.g., software agents or personal assistants, rely on a predefined and mostly constant model of the real world.

Viewing humans as situated actors means accepting that human behavior depends heavily on the actual situation in which the human is involved. Considering the above mentioned features of the real world, it should be clear that it is hardly possible to define what constitutes a situation. Furthermore, it is even difficult to define which aspects of a situation actually guide human behavior. Even if subjects are interviewed on that topic, they are hardly able to explain their own behavior. Having subjects constructing post-hoc rationalizations of their own behavior in order to give plausible explanations is a well-known problem in psychology. These considerations affect personal assistant design in so far as human behavior is somehow reflected in the events a personal assistant is able to monitor. However, this reflection is intrinsically incomplete: Computational events only reflect actions that directly affect the computer. They hardly reflect the situation guiding the user's behavior. We are con-

[4]The reader is referred to (Pfeifer & Rademakers 1991).

vinced that this is why personal assistants, e.g., Open Sesame!, experience severe difficulties in trying to detect what the behaviors guiding circumstances are and inferring when automating repetitive patterns would be beneficial for the users.

In order to take situatedness more seriously into consideration, it might be useful to support the user in acting differently in different situations. Recall the example mentioned above where Open Sesame! observed that the user always emptied the trash after dragging a document into it, but the user did not want to automate this procedure. This indicates that there are some other aspects Open Sesame! has not yet considered (and which might not be accessible at all for a personal assistant). Therefore, it might be useful to offer the user the choice between emptying or not emptying the trash. If the user fails to react within a certain amount of time, the most frequently used choice is performed automatically. Accordingly, in the context of opening a document, it might be useful to let the user choose between various editors he has been using in the past. Having preferred BBedit several times because of its capability of editing large textfiles may be obsolete when, for example, the superior formatting capabilites of another product are needed. Leaving the choice to the user might support him or her in acting situated, although this presumption should be verified in real-world situations.

Considering the discussion above, we conclude that a classical approach to personal assistant design (i.e., based on detection of behavior patterns and generalization of these patterns accross different situations) is not adequate for supporting users for the duration of the task at hand. This is partly due to the nature of the desktop environment, which is not very suitable for a "true" personal assistant, beyond automating the obvious with macros. However, the problems descibed are also due to neglect of the situatedness of human behavior. We briefly discuss a proposal for a different approach to personal assistant design in the realm of information filtering below.

Our current research interest is in information filtering, a hot topic due to electronic junk (Denning 1982) and information overload (Palme 1984). Most current approaches are based on information retrieval algorithms or machine learning techniques (Foltz & Dumais 1992). Situated Design (Pfeifer & Rademakers 1991) is a systems engineering methodology under development in our lab over the past few years focused on the "information needs" of humans involved in a particular working situation. Inspired by our experiences with Situated Design (Lueg & Müller 1996; Müller & Pfeifer to appear), we are investigating how the notion of situated action can be exploited fruitfully in the context of information filtering.

Our approach to intelligent information filtering is not based on sophisticated information retrieval algorithms or automatically acquired user models. Instead, we are investigating how users can be supported in acting situated in order to cope efficiently with information overload. As an experimental domain we chose Usenet news because it is easily accessible and well-known as testbed for information retrieval algorithms and machine learning techniques, and it is poorly structured; although there are some inherent structuring mechanisms available such as newsgroups, structured headers, etc. (Horton & Adams 1987) Nevertheless, the classification of a news article into newsgroups is solely done by the author based on his own preferences. In order to test our hypotheses, we are modifying an X-Windows based newsreader to monitor all user events and to support the user based on typical constellations in these events. So far our approach is in a different field, but similar to that of Open Sesame!.

There are several constraints to be found in the Usenet News environment that can be exploited in order to provide user support. For example, user behavior with respect to a certain thread, i.e., the number of news articles classified under the same topic, is a valuable source of information. Without rating any articles of a thread, we can conclude that a user repeatedly ignoring a thread is not interested in that particular topic. Therefore, the personal news assistant can filter this thread automatically without any interpretation of the content. Of course, it must be considered that there may be good reasons to ignore the thread at the time, e.g., time pressure of prevailing interests. Therefore, the thread is filtered but not deleted, thus enabling the user to actively influence the filtering process. The important difference of our more situated approach compared to Open Sesame!'s approach is that we avoid to generalize across situations: The user always has the possibility to influence the filtering process. Of course, the possibility to influence the assistant's behavior also holds for Open Sesame! since the user is able to turn on/off Open Sesame!'s recommendations or tune a recommendation using the menu shown in figure 4. However, while Open Sesame!'s settings are static (they do not adapt over time apart from new recommendations based newly detected behavior patterns), the filtering behavior of our newsreader continuously adapts to user behavior.

Future research based on this more situated approach to Usenet News filtering incorporates consideration of social or collaborative information filtering approaches, i.e., information filtering based on ratings

(Resnick *et al.* 1994; Shardanand & Maes 1995) or recommendations (Goldberg *et al.* 1992) given by humans instead of being computed automatically. These approaches have some common ground with our own approach in that they consider the social environment of the users. However, they do not strongly incorporate the notion of situatedness. We will then investigate whether our approach can be fruitfully exploited in other domains supposedly causing information overload such as email or WWW. Another topic to consider is that of communication overflow(Ljungberg 1996) in contrast to information overload which is a quite different but nevertheless interesting perspective.

Summary

In the previous sections, we looked at Open Sesame! as an example of a personal assistant and presented examples of other software agents for various tasks. We concluded from our experiences with Open Sesame! that, as an intelligent software agent, OpenSesame! fails to meet the mark, lacking the flexibility to support the user in acting situated. We also showed that it is almost impossible for a software agent to surmise from events and actions alone what prompts the user to make a particular decision at a given time, since the rational for the behavior is intrinsic to the current mindset of the user. We argued that instead of trying to infer what the user might do or what the user is thinking–a nearly impossible task–it would be better to bring the notion of situatedness and situated action into the design process of software agents, thus creating agents which provide active support for the task at hand without attempting to reinvent the wheel–or the user. Our Situated Design inspired approach, as discussed previously, allows us to avoid many of the pitfalls of the previous approaches by removing the necessity for user modeling and knowledge representations. Remember: "the world is its own best model!" (Rod Brooks)

Acknowledgments

Microsoft Word is copyrighted 1987-1997 by Microsoft Corporation. Open Sesame! is copyrighted 1993-1997 by Charles River Analytics (CRA). We are grateful to Charles River Analytics, especially Robin Jones and James M. Mazzu, for their help; the Swiss National Science Foundation, the Swiss National Energy Research Commission, and the Software Engineering Group (University of Zurich) for their support and help; Rolf Pfeifer for a cool research environment; Martin Müller for valuable discussions on situatedness; and Snacker for his undying devotion.

References

Armstrong, R.; Freitag, D.; Joachims, T.; and Mitchell, T. 1995. Webwatcher: A learning apprentice for the world wide web. In (**?**).

Balabanović, M., and Shoham, Y. 1995. Learning information retrieval agents: Experiments with automated web browsing. In (**?**).

Caglayan, A.; Snorrason, M.; Jacoby, J.; Mazzu, J.; and Jones, R. 1996. Lessons from open sesame! a user interface learning agent. In *Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*.

Caglayan, A. 1996. Thank you! Email to SesameBeta Mailing-List (SesameBeta@crasun.cra.com).

Carpenter, G. A., and Grossberg, S. 1987. Art-2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics* (26):4919–4930.

Charles River Analytics. 1996. Open Sesame! world wide web page http://www.opensesame.com.

Clancey, W. 1991. The frame of reference problem in the design of intelligent machines. In *Architecture for Intelligence. The 22nd Carnegie Mellon Symposium on Cognition*, 257–423.

Coen, M. H. 1994. Sodabot: A software agent environment and construction system. Technical Report A.I. Technical Report 1493, MIT Artificial Intelligence Laboratory.

Cypher, A. 1991. Eager: Programming repetitive tasks by example. In (**?**), 33–39.

Denning, P. J. 1982. Electronic junk. *Communications of the ACM* 25(3):163–165.

Etzioni, O.; Lesh, N.; and Segal, R. 1994. Building softbots for unix (preliminary report). In (**?**), 9–16.

Foltz, P. W., and Dumais, S. T. 1992. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM* 35(12):51–60.

Foner, L. N. 1993. What's an agent, anyway? a sociological case study. Technical Report Agents Memo 93-01, MIT Media Lab, Autonomous Agents Group.

General Magic, Inc. 1995. The Telescript developer environment user guide (version 1.0 alpha).

Goldberg, D.; Nichols, D.; Oki, B. M.; and Terry, D. 1992. Using collaborative filtering to weave an information tapestry. *Communications of the ACM* 35(12):61–69.

Gray, R. S.; Rus, D.; and Kotz, D. 1996. Transportable information agents. submitted to AAAI96.

Gray, R. S. 1995. Agent tcl: A transportable agent system. In (**?**).

Harnad, S. 1990. The symbol grounding problem. *Physica* D(42):335–346.

Harrison, C. G. 1995. Smart networks and intelligent agents. In *Proceedings of Mediacom '95*.

Horton, M., and Adams, R. 1987. RFC-1036 (standard for interchange of usenet messages).

Jennings, N. R., and Wooldridge, M. J. 1996. Software agents. *IEE Review* 17–20.

Kautz, H. A.; Selman, B.; Coen, M.; and Ketchpal, S. 1994. An experiment in the design of software agents. In (**?**), 43–48.

Kautz, H. A.; Selman, B.; and Coen, M. 1994. Bottom-up design of software agents. *Communications of the ACM* 37(7):143–146.

Koster, M. 1996. World wide web robots, wanderers, and spiders. *http : //info.webcrawler.com/mak/-projects/robots/robots.html*.

Lashkari, Y.; Metral, M.; and Maes, P. 1994. Collaborative interface agents. In (**?**).

Lieberman, H. 1995. Letizia: An agent that assists web browsing. In (**?**).

Ljungberg, F. 1996. An initial exploration of communication overflow. In (**?**), 19–51.

Lueg, C., and Müller, M. 1996. Cooperative systems: The right direction? In (**?**), 315–329.

Maes, P. 1994a. Agents that reduce work and information overload. *Communications of the ACM* 37(7):31–40.

Maes, P. 1994b. Social interface agents: Acquiring competence by learning from users and other agents. In (**?**), 71–78.

Markoff, J. 1994. Open sesame!: Help! there's an agent inhabiting my mac! *MacUser* 73.

Mock, K.; Lawton, L.; and Hoyle, M. A. 1996. Cyberspace game show hosts: Agents for socialization, not just entertainment. In (**?**).

Moukas, A. 1996. Amalthea: Information discovery and filtering using a multiagent evolving ecosystem. In (**?**).

Müller, M., and Pfeifer, R. to appear. *Developing Effective Computer Systems Supporting Knowledge Intensive Work: Situated Design in a Paper Mill*. In K. Mehdi and J. Liebowitz, eds. *Cases in Information Technology Management in Modern Organizations*.

Norman, D. 1994. How people might interact with agents. *Communications of the ACM* 37(7):68–71.

Nwana, H. S. 1996. Software agents: An overview. *Knowledge Engineering Review*.

Palme, J. 1984. You have 134 unread mail! Do you want to read them now? In (**?**), 175–184.

Pfeifer, R., and Rademakers, P. 1991. Situated adaptive design: Toward a methodology for knowledge systems development. In (**?**), 53–64.

Pylyshyn, Z., ed. 1988. *The Robot's Dilemma. The Frame Problem in Artificial Intelligence*. Ablex.

Rao, A. S., and Georgeff, M. P. 1995. BDI agents: from theory to practice. In (**?**), 312–319.

Rasmus, D. W. 1995. Intelligent agents: Dai goes to work. *PC AI* 27.

Resnick, P.; Iacovou, N.; Suchak, M.; Berstrom, P.; and Riedl, J. 1994. GroupLens: An open architecture for collaborative filtering of netnews. In (**?**), 175–186.

Shardanand, U., and Maes, P. 1995. Social information filtering: Algorithms for automating "word of the mouth". In (**?**), 210–217.

Shoham, Y. 1993. Agent-oriented programming. *Artificial Intelligence* (60):51–92.

Suchman, L. 1987. *Plans and situated action - The Problem of Human-Machine Communication*. Cambridge University Press.

Sun Microsystems, Inc. 1996. Documentation Java(tm) and JavaSoft(tm) products.

Tambe, M.; Johnson, W.; Jones, R. M.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine* (Spring):15–39.

Wooldridge, M. J., and Jennings, N. R., eds. 1995a. *Intelligent Agents*. Lecture Notes in Artificial Intelligence. Springer Verlag. Proceedings of the ECAI-94 Workshop on Agent Theories, Architectures, and Languages.

Wooldridge, M. J., and Jennings, N. R. 1995b. Intelligent agents: Theory and practice. *The Knowledge Engineering Review* 10(2):115–152.