Active Interfaces through Software Agents

Amedeo Cesta [*] and Daniela D'Aloisi [**]

[*] IP-CNR, National Research Council of Italy

Viale Marx 15, I-00137 Rome, Italy

amedeo@pscs2.irmkant.rm.cnr.it

[**] Fondazione Ugo Bordoni

Via B. Castiglione 59, I-00142 Rome, Italy

dany@fub.it

Abstract

This paper concerns the development of an interface environment to help users in repetitive tasks in office work. The main ideas in the project concern: the development of active interfaces that autonomously perform tasks minimizing the interaction with the user; the use of the agent-oriented paradigm to provide both distributedness and incrementality of the software environment. The paper quickly illustrates the main issues addressed in the project and shows how they are exploited in the development of an active interface for filtering e-mail messages. The architecture of the filtering agent follows a multi-agent implementation.

1 Interfacing Useful Software Tools

The spreading of electronic instruments is getting common people more and more in touch with software tools not always easy to be handled with. Particular emphasis can be put on network tools and information sources and on their use for different services in both domestic and office contexts.

Due to the world-wide diffusion of computer networks, to the daily increasing number of users connected to them and to the quantity of services offered, it seems that a lot of our engagements and amusements will be satisfied through the networks while we are comfortably sitting on our chair. This scenario is likely to become true only if interfaces designed to make acceptable and easy using these offered tools are supplied. In some way, the employment of such tools should be user-transparent.

In fact people could not immediately accept the introduction of *machines* the utility of which is not so clear since they change completely their way of facing (and solving) problems. Also when the users are competent in dealing with sophisticated software mechanisms, the context can change but the problems are similar. In fact, high-skilled people do not disdain an aid that would make their work easy, for example by saving their time. In fact, the full exploitation of services and data available through the Internet takes a lot of time, time that is stolen to their job.

A different issue concerns the usually large amount of knowledge needed to actually use those tools. Moreover, since the number of different versions or releases increases very

fast, and it is difficult to follows all the changes especially if the acquisition of any competence is hard.
Both in novice and expert cases, and for all the possible intermediate cases, the interfaces with these tools have the great responsibility of making the access easier. If the use of computers must be really enlarged to non-specialist people, their management should be easy and the interface should take care to do something more than what the users strictly request. In fact, since the goal is to disengage the user as far as it is possible from learning the tool, the interface should be *active*, i.e., it should support the user in each phase of her interaction. It should suggest solutions and have deductive capabilities.
In our project we are considering interfaces for useful software tools available on the network. A lot of utilities are accessible on public domain repositories which are useful to solve particular problems but also: (a) they are very powerful but not easily comprehensible. So if from one hand they save time in small tasks, on the other hand they require a lot of time to be learned; (b) most of the utilities have programmer-oriented interfaces and, as a consequence, they are mainly used by "hackers" or specialized programmers.
Our work has been focussed on building interfaces that avoid users to learn much about a particular tool. In addressing this kind of problem we have restricted the application context to the development of interfaces for tedious and repetitive office tasks, as dealing with e-mail messages or scheduling meetings or looking for useful information on the network. These tasks take advantage from or are motivated by the existence of the network, but the wide use and success of tools solving them are strictly depended on their conjunction with effective and supportive interfaces.
In the project we follows two methodological goals: (a) building active interfaces that carry out some additional role with respect to being simple filters between users and applications. The concept of active interface has been introduced in several research in human-computer interaction (e.g., [5, 7]); (b) employing the agent-oriented paradigm to implement the interfaces in order to make the whole system both easily extensible and tailorable to single users. A number of interesting work is going on similar issues (e.g., [3, 6]).
In the rest of the paper we show an example of interface to a software tool that avoids users to learn a specific language; has a supporting role with respect to the user; follows an agent-oriented implementation that allows for the specification of single parts of the whole system.

2 Active Management of E-mail Messages

The project we are involved has the aim of incrementally developing a software Interface Agent connecting the user with an intelligent distributed architecture that supports and assists her in dealing with disparate tasks, tackling different network utilities and utilizing office services. We have built several specialized agents in order to increase the efficiency of the user's daily activities. Task of the Interface Agent is to select the agent(s) that can satisfy the user's needs, to coordinate the work of different agents, to make them cooperate in presence of a common goal, to communicate with the external world be it represented either by intelligent or by non-intelligent entities. Interface Agents acting on the behalf of different users can interact with each other in order to get help and can cooperate among them to accomplish particular goals.
At the present, the architecture consists of: a first implementation of the Interface Agent,

a Mail Agent, a Meeting Agent and an Info Agent. Each agent consists in turn of sub-agents specialized to solve a particular problem. In this paper we illustrate the architecture and the behavior of the Mail Agent in order to show how the concepts above illustrated are translated into an operative framework. The *Mail Agent* is in charge of handling the incoming e-mail messages according to a user's profile. With e-mail messages we mean every types of electronic mail such as bulletin-boards, mailing-lists, interest groups, etc. To cope with them specific tools exist, e.g., packages like Procmail or MailAgent, that allow a user to filter the incoming messages: such tools are powerful but of difficult usage for non-expert programmers. Our *Mail Agent* hides to the users the use of the Procmail software and adds an active task by automatically suggesting modifications in the profile on the basis of a comparison between accepted and rejected messages.

Procmail is a software that checks each incoming e-mail message according to a number of user-defined filters. Filters have to be specified using the Procmail command language consisting of a set of operators, as in the following example:

```
MAILDIR=/usr/users/amedeo/mail
:0 H
         * ^ From.*dai-list
         :0 c
         $MAILDIR/bboards/dai
         :0
         ! dany@fub.it
```

This command states that any message coming from `dai-list` should be stored in the user's sub-directory `/bboards/dai`, and then forwarded to `dany@fub.it`. The command language becomes more and more complex as more sophisticated performances are required.

A first version of the *Mail Agent* was developed addressing the specific problem of avoiding the user to learn the language: a high level language that synthesized Procmail commands was designed and then an interface was implemented that made such a language easily accessible by a window interface. The aim was to avoid the user to master the tool language. The solution required a serious amount of work in order to write an interpreter for graphical specification into Procmail commands, but turned out being inadequate because of its rigidity.

While this first solution represented a considerable effort in addressing the problem of the knowledge about the tool language and its use, it completely neglected how to acquire competence in an effective use of the tools. This issue is important and involves the role of the programmer's skill and experience: in fact, in order to effectively use Procmail, the user should reach a good level of expertise and gather a certain amount of knowledge about the possibilities offered by the tool.

The second version follows the general requirements of our project and collects the results from the previous system. In particular, it takes the following issues into account: (a) *Agent-based design:* the development of the interface follows the agent metaphor. The job of e-mail filtering is carried out by an intelligent agent able to interact with the user and with the external world, able to take the initiative when required, and able to reasoning about the user and her needs. (b) *Distributed architecture:* the *Mail Agent* consists of several sub-agents each performing a particular set of actions. The sub-agents cooperate among them to accomplish the required final task. (c) *Integration of public-domain tools:*

some of the sub-agents perform their jobs by using public domain software, i.e., Procmail and WAIS. (d) *Active behavior:* the *Mail Agent* is able to improve its performance by extracting information from the past history of its interaction with the user.

2.1 The Mail Agent Structure

The *Mail Agent* handles e-mail messages on user's behalf: in order to avoid the overflow of information, it helps the user to automatically select among the incoming e-mail messages those satisfying constraints specified in one or more profiles. Only the messages that correspond to such profiles are put in the user's mail box, while the other are rejected. Actually these messages are not throwing away but are used to suggest possible modifications in the filter setting. In fact, the rejected messages are indexed and then compared with the accepted ones to verify if similarity exists, i.e., whether a rejected message can be caught again since it contains information that the user could find useful. This allows the user to decide a modification for the filter setting following a *Mail Agent*'s suggestion. This is part of the active behavior of the interface that tries to go beyond requests explicitly done.
As above mentioned, the architecture of the MailAgent is distributed. It consists of four sub-agents each devoted to deal with a particular issue in the context of filtering e-mail messages: the *Preference Specificator*, the *Refinement Proposer*, the *ProcMail Agent* and the *Message Manager*. Figure 1 shows the components of the *MailAgent* and the information flow.

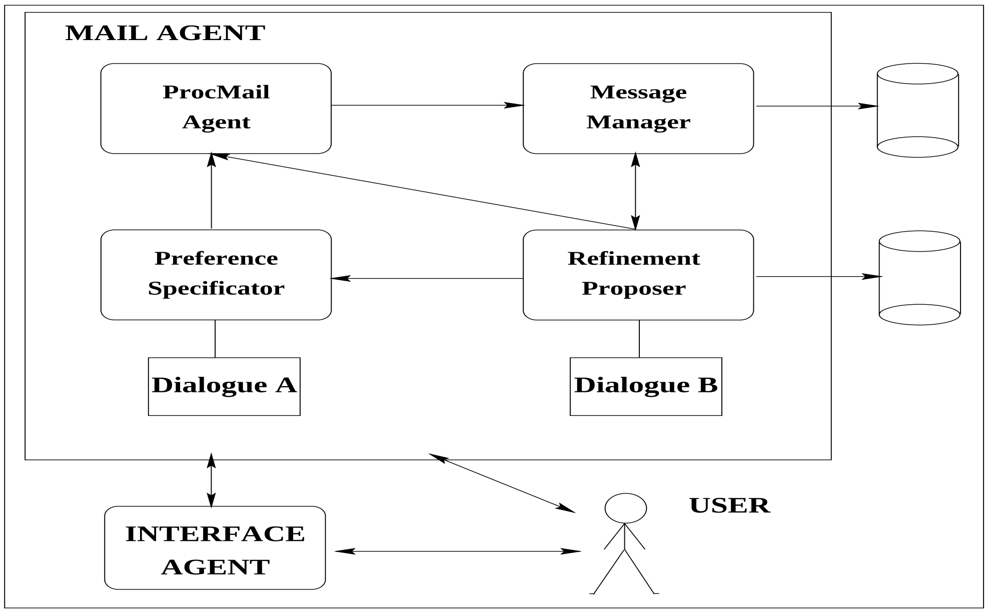

Figure 1: The architecture of the *Mail Agent*

The choice of implementing more entities is due to flexibility and efficiency criteria. Moreover, each agent should be though of as an instance of an agent type that could be active in different contexts. Each sub-agent has precise tasks and utilizes software functionality to accomplish them: they cooperate among them basically using the KQML language [4] to carry out the final task of filtering the e-mail messages of the user.

The *Interface Agent* receives a request from the user of being connected with the *Mail Agent*: the request can be either direct, e.g., the user selects just this agent from a menu, or indirect, e.g., the agent deduces the user's need from the current state of affairs. Once the request has been interpreted, the *Interface Agent* connects the user directly with the *Mail Agent* so that its architecture is transparent to the user: after the contact is established, the user interacts directly with the *Mail Agent*.
The *Preference Specificator* and the *Refinement Proposer* need to interact with the user since they deal with her interest profile: at the present, they show a window the user fills in but we are working at a new version in which this information is synthesized by the *Interface Agent* on the ground of the data it owns.
The *Preference Specificator* agent assists the user in specifying the filter's constraints and then translates these preferences for the *ProcMail Agent*. In the current version, the agent dialogues directly with the user through a dialogue window in which she can type the constraints to be satisfied by a message in order to be selected among those contained in the incoming mail. The *Preference Specificator* also translates the descriptions into an intermediate format that becomes the body of a KQML message to be sent to the *ProcMail agent*.
The *ProcMail Agent* is in charge of generating Procmail filters. Along with the *Preference Specificator*, it avoids the user to learn the command language of ProcMail: in fact the user can specify her preferences in a plain mode without being aware of the complexity of the target language. The first of its tasks is to translate the information coming from the *Preference Specificator* and asks to the ProcMail program, that is part of its body, to synthesize the filter. Then the filter returned by ProcMail is applied on the incoming e-mail messages. Each message is marked as accepted or rejected according to the filter(s), and then passed to the *Message Manager*. The accepted messages are also put in the mail box of the user.
The key role for the active part of the interface is played by the combined activity of the *Preference Specificator* and the *Refinement Proposer*. In particular:

- The main task of the *Message Manager* agent is to handle with the sets of the accepted and rejected messages in order to discover whether some of the rejected messages is similar to one or more of the accepted messages. In an affirmative case, some message could be recovered and proposed to the user: then that can be used to propose a revision of the filters. The comparison between the messages is performed by WAIS (Wide Area Information Server), a public-domain tool specialized in querying databases by employing information retrieval techniques. Each rejected message is transformed by WAIS in a query against the indexed content of *DB Mail*. If relevant messages are found in the rejected set then they are sent to the *Refinement Proposer*.

- Task of the *Refinement Proposer* is to extract constraints for new filters from the analysis of the relevant messages selected by the *Message Manager*.
  The messages coming from the rejected set and selected by the *Message Manager* are stored by the *Refinement Proposer* in a database, called *DB Relevant*. These messages are shown to the user. For each message, the *Refinement Proposer* creates a new filter deducing it from the features of the message most similar to it: also the message is shown with enlightened the relevant keywords. The user can choose to accept or to modify or to reject the suggested filter. If the user accepts some of

the modification proposed, this monitoring module sends a request to the *ProcMail Agent* of revising the current set of filters.

The idea is that, given a standard generation of filters by the user, the system gets a first idea of the user interests and starts a work of observation and revision. Some of the rejected messages may be "similar" to one or more of the accepted ones. Chances are that the messages were deleted because of a "rigid" use of ProcMail by the user. A negotiation phase with the user is started to understand if he acknowledges the acceptance of the retrieved similar messages.
The system described is implemented on a Sun workstation and is currently used by several people in our office environment.

3 Conclusions

This paper shortly describes the general ideas we are pursuing in building up intelligent interfaces to help users in office work. The main methodological issues are the development of active and agent-based interfaces. In the particular example shown here, the *Mail Agent*, it is possible to see the application of both those aspects. It is worth noting the active role introduced by adding to the system the *Message Manager* and the *Refinement Proposer* which are responsible for suggesting some improvements with respect to the raw proposal initially set by the user through the preference specification. A detailed description of the *Mail Agent* can be found in [1], while an extensive description of our project is [2].

Acknowledgments

The authors thank Vittorio Giannini for the implemention of the mail agent and for stimulation discussions on the topic. Daniela D'Aloisi carried out their work in the framework of the agreement between the FUB and the Italian PT Administration. Amedeo Cesta's work is realized in the framework of the ESPRIT III Working Group No.8319 "A Common Formal Model of Cooperating Intelligent Agents (ModelAge)".

References

[1] Cesta, A., D'Aloisi, D., Giannini, V., Active Interfaces for Software Tools. In Y.Anzai, K.Ogawa and H.Mori (editors), *Symbiosis of Human and Artifact* (Proceedings of the 6th International Conference on Human-Computer Interaction, Tokyo, July 9-14, 1995), Elsevier Science B.V., 1995, pp.225-230.

[2] Cesta, A., D'Aloisi, D., Implementing Active Interfaces through Software Agents. Tech. Rep. IP-CNR, September 1995.

[3] Etzioni, O., Levy, H.M., Segal, R.B., Thekkath, C.A., The Softbot Approach to OS Interfaces. *IEEE Software*, July, 1995, pp.42-51.

[4] Finin, T., Weber, J., et al., Specification of the KQML Agent-Communication Language. DRAFT, February 1994.

[5] Fisher, G., Lemke, A. and Schwab, T., 1985. Knowledge-Based Help Systems. Human Factors in Computing Systems. *CHI '85 Conference Proceedings*, San Francisco, CA, ACM, 1985, pp.161-167.

[6] Genesereth, M.R., Ketchpel, S.F., Software Agents. *Communication of ACM*, Special Number on Software Agents, July 1994, pp.48-53.

[7] Laurel, B. (ed.),*The Art of Human-Computer Interface Design.* Addison-Wesley, 1990.