Desktop Agent Manager (DAM): Decision Mechanism

Deborah Ann Guerrera Ercolini
The MITRE Corporation

Mieczyslaw M. Kokar
Northeastern University

Address for correspondence:
Mieczyslaw M. Kokar
Department of Electrical and Computer Engineering
Northeastern University
360 Huntington Avenue
Boston, MA 02115
phone: (617) 373-4849
fax: (617) 373-2501
email: kokar@coe.neu.edu

Abstract

Desktop software agents are background processes that notify a computer user of certain predefined events. The complexity of desktop computing is increasing due to the proliferation of software agents. A Desktop Agent Manager (DAM) may become an important component of desktop computing. This paper focuses on the decision mechanism for the DAM to decide which agents should be allowed to access the user, which agents' results should be stored for future review by the user, and which of the agents should be filtered out. We prove the validity of the concept of a DAM by developing an architecture that includes both a prototype of the Desktop Agent Manager and a simulator of various types of agents. In this paper we present the results of our simulations and analyses of the DAM decision mechanism.

1. Introduction

There are many applications that reside on the typical office computer: electronic mail, word processor, spreadsheet, personal database manager, corporate database manager, scheduler, etc. Many COTS (components of the shelf) have *software agents* as components. Typically, a software agent is defined as an autonomous background process that communicates with both the user and other programs. Sometimes agents can be configured by end users (the application provides a simple user interface for the user to describe their needs), and at other times agents need to be developed by system programmers.

With the increased utilization of software agents comes distraction. Agents interrupt workflow as they query for user feedback. The interruptions can be *random*, *scheduled*, *ad hoc*, or *triggered*. These events/interruptions can happen while the user is writing a document, watching a class or lecture (via video on desktop), or participating in electronic meeting (via screen sharing, VTC). This paper addresses the solution to alleviate the problem of many agents interrupting the user's workflow.

Complex applications on today's computer desktop require communication, coordination, and collaboration between the user and tools such as:

- shared screens: users can view and manipulate each others computer desktop;
- shared folders: users can share repositories;
- desktop video conferencing: users can have face-to-face meeting from using their computers;
- public calendars: users can view and manipulate other users' calendars.

Consider the following events that could be produced by various agents and appear on the desktop at any one time:

- Notifier (your automatic query is finished; your 10:00 am meeting has been changed to 11:00; e-mail is in your in-basket);
- Workflow Notifier (sign this purchase requisition within 24 hours; review this document by 2/22/94);
- Yellow/Red Flag Notifier (your budget is overrunning; schedule is slipping);
- Alarm (printer is out of paper, final report is a month late);
- Urgent request via DVTC (prepare a one page write-up for annual report on collaborative computing for specialty group);
- Network Load Monitor (load is low);
- Pseudo-monitor (you may have CNN running in the background to watch the late breaking news stories, weather, etc.).

There are many examples of commercial-off-the-shelf (COTS) packages which incorporate agents as components of their applications. For example, BeyondMail (Miley, 1993), an electronic mail system, filters mail as it is received. BeyondMail can filter messages by the sender of the message, by the subject, or by certain keywords in the body of the message. The user can specify to filter the messages into a particular folder or to automatically forward a message to someone else. Open Sesame! (Streeter, 1993), a software application for the Macintosh from Charles River Analytics Inc., provides intelligent agents which mimic the user's repetitive general desktop movements, such as opening and closing applications and files. This product uses neural networks and expert-system knowledge bases to achieve this performance. Magnet from No Hands Software (Tessler, 1993) is a software package

which incorporates intelligent agents for searching files on Macintosh network systems. These agents can be scheduled to run at specific times or can be triggered by specific events.

There are at least two possible strategies that can be used in the solution to the problem of coordination of multiple agents communicating with the user (Bird, 1993): communication-based (distributed) or centralized. In the distributed approach, particular agents would have to come to an agreement on who should be allowed to access the user and who should not, through a rather extensive communication process. There are at least two problems with this kind of approach. First, agents developed by many vendors do not have a common standard communication protocol, and therefore to implement this approach, either a new standard would have to be imposed, or interfaces would have to be developed for all the kinds of agents. Second, for the user to be able to interact with all the agents in a similar manner, all the agents would have to have a common mechanism that the user could use to establish a user access control policy. For these two reasons, the distributed solution is very difficult to implement. More discussion of distributed heterogeneous agents can be found in (Adler et al., 1992). In the centralized approach, one agent would have to be designated as a coordinator of all other agents. The user would need to interact only with this agent to establish or change the user-access policy. This seems to be a more feasible approach to the coordination of multiple agents. For this reason, in our research we focused on such a central agent coordinator, which we call Desktop Agent Manager (DAM). Our focus is on heterogeneous agents working on unrelated tasks and therefore communication among agents is of a lesser importance. The need for a user interface agent was also identified in (Avouris et al., 1993) for a collection of homogeneous agents working on a common task.

The goal of our research (Ercolini, 1994) was to investigate the concept of a desktop agent manager. Towards this goal, we first performed a literature search into the existing solutions. A brief overview of

existing solutions is presented in Section 2. Our first objective was to understand the requirements for such an agent manager. Our second objective was to find out whether either a solution that implements our requirements already exists, or identify such solutions that could be modified to fulfill the DAM requirements. Since none of the existing solutions satisfied our requirements, we decided to develop a DAM prototype. For this we had to define all the necessary functional components of a DAM, the architecture of the DAM and the communication protocol to communicate with the user and with the managed agents. Additionally, we had to develop an environment to simulate various agents. From the user's point of view, the main functional element of the DAM is the filtering mechanism which makes a decision on whether an agent should report to the user or not. The decision mechanism proposed in (Ercolini, 1994) is described in Section 3. The whole architecture is described in Section 4. One of the major tasks in this research was to evaluate the decision mechanism. The simulation is described in Section 5. The results of simulations with the DAM are described in Section 6. In Section 7 we present the analysis of the simulation results. Section 8 contains conclusions and suggestions for further research in this area.

2. Existing Solutions

There are many ongoing research efforts focused on software agents. Some efforts are focused on the architecture of the agents, some are focused on how agents interact, while others focus on how agents are made to be intelligent. No solutions were found that addressed how to manage software agents for the user in the manner proposed. However, several related research efforts could potentially provide solutions if additional functionality or investigation was done.

The Envoy Framework (Palaniappan et al, 1992) developed at Brown University addressed the need for software agents (envoys) to work with the existing "envoy-aware" agents. As part of this framework, two management components were developed. A "Bureau Chief" manages envoys; it keeps track of user's envoys, creates new envoys, assigns new tasks to envoys, and keeps track of envoy-aware applications. A "Mission Summary" is the user interface to Bureau Chief and envoys. It displays the status of the envoys, displays reports from the envoys, and allows the user to cancel/stop an envoy. Either of these components could potentially be modified to provide a solution to managing the agents. Our research can be viewed as an extension to the Envoy Framework research. Our DAM is very similar to Mission Summary, in that it keeps track of the existing agents and what tasks they are performing. Envoy research indicated that users wanted a more sophisticated Mission Summary. Our solution takes Mission Summary manager one step further. The Desktop Agent Manager has a decision mechanism to decide whether an agent can display its results to the user.

The Object Management Group (OMG) is providing an architecture for distributed, cross-platform application communication. OMG has defined the standard called the Common Object Request Broker Architecture (CORBA) model. The CORBA model defines the language and a set of services which can be used to define and establish interagent communication. A manager of agents could communicate with its agents and with the user using a CORBA compliant system, but CORBA does not define any specifics that are directly related to the issue of agent management.

CUBRICON Intelligent Window Manager (CIWM) (Funke et al., 1993) is a prototype which defines where particular types of windows (such as text, graphical map, table, and form windows) should be placed upon opening. Some windows are related to others and therefore need to be placed in close proximity. Some windows have more importance than others. If there is no more screen space for

another window, the system has to decide which of the open windows should be closed. In designing our solution we considered similar variables as described in (Funke et al., 1993): priority of the agent; importance of the results of the agent both with keywords and with prioritized results; when the agent was initiated; given a time limit on the agent, how late is the agent; what is the current user status, etc.

Research presented in (Zlotkin and Rosenschein, 1993) addresses the issue of fairness in interaction mechanisms among agents. Without fairness, some agents can take advantage of others and the efficiency of the system will suffer. We benefited from the research presented in (Zlotkin and Rosenschein, 1993) by incorporating the results into the design of the decision mechanism of our DAM.

Research reported in (Maes & Kozierok, 1993), (Kozierok and Maes, 1993) and (Sheth & Maes, 1993) focuses on using machine learning techniques to develop intelligent agents, rather than have the user program the agents. The authors claim that machine learning techniques can only be used in systems where there is repetitive behavior (without it, agents will not be able to learn) and where the repetitive behavior differs depending on the user (if all users were the same, then there would only be one environment to learn). The agents can learn by observing users. These agents use memory-based learning to learn the situation-action pairs. When a situation happens, the agent searches for similar situations. When one is found, a confidence level is used to either suggest an action or to do an action. When a suggested action is declined, the agent receives negative feedback. When agents become "smart", they start interacting with the user, and when they become "intelligent", they start performing some tasks for the user. The machine learning approach is very natural for an advanced Desktop Agent Manager. So far, we have implemented a "smart" manager in that it decides whether to report the results of various agents to the user.

3. Decision Mechanism

The principal components of the system in which the Desktop Agent Manager would operate are the user and the multiple heterogeneous agents. The following scenario is an example of interactions of the Desktop Agent Manager with these components. When an agent is created, the application has the ability to register the agent with the Desktop Agent Manager. The agent reports to the Desktop Agent Manager before notifying the user. The Desktop Agent Manager decides whether the agent is important enough to directly report the results to the user or the agent should be "filtered" by the DAM. The Desktop Agent Manager returns one of two results, *do not filter agent*, i.e., report directly to the user or *filter agent*, i.e., do not report to the user. There may be a case where the user wants to quickly see the results and then delve into them later, so the user should be able to specify that both actions take place, in other words, the user should be notified immediately and the Desktop Agent Manager should also have a log of the situation.

In order to implement this scenario, the Desktop Agent Manager must have the following functionality: (1) manage the different types of agents, (2) collect and manage requirements from the user, (3) keep track of the data that the agents provide to the user and display the results of the agents in an intuitive manner.

Since our primary concern is with the decision mechanism of the DAM, in the rest of this section we discuss the first of these functions. The decision mechanism of the DAM must be based on parameters representing both the user requirements and the agents that try to access the user. Since the decision mechanism is to be generic, appropriate for various agents, the parameters must be obtainable from all the agents. The following parameters have been selected based upon the review

of various kinds of agents (numbers in parentheses represent numerical values of the variables that are used in the computation of the importance of the agent):

- *user status: bored* (10)*, flexible* (20), *busy* (30), *do-not-disturb* (50);
- *agent priority,* as defined by the application: *low* (1), *medium* (2), *high* (3);
- *keyword association level* for a keyword found in the agent's result: *general* (2), *associated with the agent category* (3), *agent specific* (4);
- *keyword priority: low* (1), *medium* (2), *high* (3);
- *time limit* (in minutes) for processing agent's task as specified by the application (optional);
- *priority of the result* (optional): *low* (1), *medium* (2), *high* (3);

User status is a parameter that the user of the DAM can vary depending on how much responsibility the user wants to delegate to the DAM. *Agent priority* represents the priority of a specific application. The user will be able to specify keywords as part of the profile. The user can associate *keywords* at different *keyword association levels*. A keyword can be associated with a specific agent, to allow all instances of that agent to be associated with this keyword. Keywords can also be associated at the agent category level. This allows keywords to be associated with any agent that falls into a particular category. Lastly, a keyword can be associated with all agents.

The *importance level* of the agent calculated by the decision mechanism will be higher for any agent reporting results which contain the defined keywords in either the *message* or the *brief report*. The weight of the keyword association will be higher for the more specific level of

association. Keywords can also be assigned *keyword priority* levels, which will be used in calculating the importance of an agent.

Agent process *time limit* is an optional parameter that an agent can require the user to specify. It is the time duration of processing an agent's task. The importance of the agent is increased if the agent is early and decreased if the agent is overdue with respect to this time limit.

When reporting its results, the agent has the option to include a *priority of the result* that it gathered. This option allows agents which can evaluate their own results pass this information to the DAM to use in its decision making process.

Using the above variables, the procedure presented in Figure 1 is used to calculate the importance value of the agent.

4. DAM Architecture

To evaluate and verify the concept of the Desktop Agent Manager, a prototype of the DAM has been developed. Since agents are required to interact with the Desktop Agent Manager, rather than interfacing directly to the agents that are available today, we implemented a simulated environment in which various kinds of agents can be simulated. The benefit of this approach is one common interface for all the agents. The major components of our architecture (the object model) are shown in Figure 2. A more detailed description of the whole system can be found in (Ercolini, 1994).

The DAM is represented in the bottom part of the figure. The upper part shows the simulator, the simulated agent, the repository of simulated agents and the user. The user can interact with the simulator, the simulated agent and the DAM.

The Desktop Agent Manager has a *User Profile*, a *Decision Maker*, and an *Agent Record* as associated objects. The Simulator is the object that controls the simulation environment. The *Simulated Agent* is the object that represents the external agents of the system. The *Generic Agent* (not shown in Figure 2) and all of its subclasses represent the different types of simulated agents and is the repository for the canned results of the simulated agents. Figure 2 is an overview of the entire object model with the *Agent Representation* subject collapsed into a simple list of the objects contained in the subject.

Once the agent is created, it registers with the Desktop Agent Manager. The agent must provide the DAM with its name, its priority as assigned by the user, the agent category that it belongs to (if no category is passed, then the default value is 'Unknown'), and user desired result time limit. The DAM returns a unique identifier to the agent, that is used when reporting results or status change to the DAM. Once registered, the agent begins processing its task. When its task is completed, it sends its results to the DAM for approval.

Following the specs of the Envoy Framework (Palaniappan et al, 1992), the agent must provide a simple message to the user. The agent can also provide either a brief report to the DAM, which contains a more detailed explanation than the message, or a detailed report in the form of a file with the results of the task. The agent must also provide the application which is invoked for the user to view the results.

This information can either be provided to the DAM at the time of registering or at the time of reporting results. This flexibility allows for a one-time setup if the agent always uses the same application for reporting its results as well as for an agent to be able to return results from many different applications. Another parameter the agent may report is the priority of the results of its task. Some software agents understand what they are reporting and can judge the importance of the results.

When the DAM responds, the Simulated Agent receives a decision. If the decision is “do not filter agent”, then the Simulated Agent can report its results to the user. If the decision is “filter agent”, the Simulated Agent must not report its results to the user. In either case, the Simulated Agent ends its task.

On request, the DAM will provide the user with the information of the currently stored agents. The DAM stores the time at which the agent was created, the name of the agent, and category of the agent. The DAM also stores the message, the brief report, and the detailed report of the agent (if applicable). The user is able to access the detailed report in the form of the native application of the data. The DAM invokes this application with the appropriate data at the request of the user. The DAM also stores what its own decision was when the agent reported, i.e. whether or not the agent was filtered. The user is able to delete agents from the stored agent information, to sort the view of the agents on several areas, like priority, category, name, time created, status, etc.

The simulated agent may report a change in status to the DAM. This function should not be used to change its status to "Completed". The agent may update its status with the following values: "Working", "Stopped", and "Error". To do this the agent must also provide the unique identifier

that the DAM returned when the agent registered. An optional parameter to this function is a simple message which might explain why the change in status occurred. Examples of the simple message for the "Stopped" status value could contain by whom and when (what ever has been defined by the agent) and for the "Error" status values could contain a reason for the error. The agent should not expect a return value from the DAM in response to status change.

The agent is the initiator of communication with the DAM. If the agent does not receive a response, then the agent assumes that there is a problem and continues processing. When it finishes its task, the agent reports to the user as it normally would do. When reporting to the DAM, if the agent does not have a response from the DAM, it reports directly to the user assuming that the DAM is no longer functional.

5. Description of Simulation

5.1. Simulation Environment Description

The Simulation Environment has been incorporated into this development as a mechanism to generate and gather data and to simulate how the DAM might act if actually put in production mode. The user of the simulation environment is the system evaluator. The simulation environment includes the components (ref. Figure 2): *Simulated Agents*, a simulation controller *(Simulator),* and a simulated agent repository *(Agent Representation).* The Simulated Agent was described in Section 3.

The Simulator allows the user to automatically generate random agents, vary the rate of simulation, and create ad-hoc agents. In the automatic generation mode, agents (names and types)

are selected randomly from the available agents in the simulated agent repository. Each agent is provided a random processing time and random priority. Random canned results and scenarios are also selected from a pool of agent scenarios in the repository. This information repository contains data files and associated applications so that the agent can display different types of data. Later in this paper we show examples of various types of agents and associated information stored in the repository.

When creating ad-hoc agents, the system evaluator has to specify the name of the agent. If the agent is already defined in the system, the simulation environment provides the type of agent. If the agent created is not in the predefined set, then the type of agent is automatically 'Unknown'. The user needs to provide the priority and, if desired/applicable, the time limit desired on the return of the results. Like with the automatic generation of agents, the simulator provides the canned results, including the simple message and the brief or detailed reports, as applicable, and the approximate length of time that the agent takes to complete its task.

5.2. Simulated Agents

There are many different types of agents and many different ways in which to categorize these agents. One categorization of agents is by how or where they are invoked. There are agents that are externally invoked *(external)*. These agents are generated when another user requests them. This is seen in applications today like Timbuktu, email and Desktop VTC. In these cases, some external entity is invoking an interruption. The DAM must be able to intercept these types of agents, if necessary. The other type of agent is *internally invoked*. These kind can be further classified into two categories. *Single task* agents refers to the type of agent that is created once to

do a specific task and once it is done, it is destroyed. The other type is a kind of agent that does its task over and over, referred to as a *Re-occurring* agent. This can be a *scheduled* query agent or an agent which is *monitoring* for a specific event and when the event occurs, the agent is activated. In our system the software agents defined in Table 1 have been simulated. They are briefly described using the types of categories described above.

6. Testing Results

For testing purposes, three sets of simulated agents were generated randomly using the Simulator's Control Panel. Using the same agent sets with different environmental conditions provides a common reference frame to the testing. The three scenarios of simulated agents that were used in the testing of the simulation are described in detail in (Ercolini, 1994). These scenarios were used at each rate of the simulation with and without the DAM activated. There was also a predefined keyword set used during the testing of the system. All other decision mechanism variables were varied randomly on all levels.

In the following figures we present the results of our simulations. Figures 3 and 4 present the raw data collected when running the three test scenarios. These figures show the default threshold values of the user status as compared to the importance values calculated for the agents. Any agent with a value that falls on or above the threshold value will not be filtered by the Desktop Agent Manager, i.e. the agent can report its results directly to the user. The first figure is the importance values calculated without using the predefined keywords, and the second figure is the importance values calculated when using the predefined set of keywords.

7. Analysis of Results

Although our overall goal was to investigate the whole concept of a desktop agent manager, including the specifications, the design and the user interface, in this paper we focus only on the decision mechanism of the DAM. Our primary interest was to investigate the "effectiveness" of the DAM with respect to its decisions on which agents and under what conditions should be filtered out and not allowed to access the user, and when they should be allowed to present their results to the user directly. While the effectiveness of the decision mechanism can be defined as the number of agents not filtered by the decision mechanism, the real questions are (1) whether the user agrees with the DAM's decisions on which of the agents should be filtered out and which should not, and (2) whether the DAM's global policy on the number of agents that are filtered out is well synchronized with the user as represented by the User Status variable, i.e., whether the user will be able to control the strictness of the filtering mechanism through this variable. To answer the first of the above questions, a human factors study is needed, including real users in real scenarios. This kind of study is beyond the scope of this paper. We concentrated on the second question, i.e., on the relation between the user status and the number of filtered agents.

To assess the validity of the agent's global policy we need to understand what kind of relation should we expect, what kind of relation is "good", or "correct"? In our investigations we assumed that the characteristic of the DAM in terms of the number of agents that are filtered should be "linear" with respect to the *user status* value. We write linear in quotes since *user status* is not a quantitative variable. Nevertheless, it is an ordinal-scale variable. Intuitively, the linearity here means that if *user status* changes by one unit, for instance from *bored* to *flexible*, the increase in the number of filtered agents should be the same as when *user status* changes from *busy* to *do not*

disturb. It also means that the number of agent interruptions should decrease as the user status level increases.

Figures 5 and 6 show the percentage of agents which were not filtered by the Desktop Agent Manager versus the *user status* level. Figure 5 is for the agents running without using a keyword set and the Figure 6 is for the agents using the keyword set.
As expected, percentage of agents not filtered increases monotonically as the *user status* level moves from *do-not-disturb* to *bored*, approaching the number of agents generated. At the *do-not-disturb* level, a small percentage of agents are not filtered and at the *bored* level, all agents are not filtered. *User status* levels *busy* and *bored* fall in between these extremes roughly linearly. It can be concluded that the decision mechanism filters agents correctly with respect to the *user status* level.

In analyzing how the keywords affect the decision mechanism, using Figures 5 and 6, we can see that although the keyword capability of the decision mechanism increases the importance of agents, the increase in importance values is not constant across all agents and that keywords do not make a predictable impact. The keyword importance values increased sensitivity slightly, but not enough to conclude that they have a significant weighting in the decision mechanism.

In order to better understand the effects of the decision mechanism we analyzed the distribution of the importance values. Figures 7 and 8 show the distribution of the importance values calculated for all the agents for the test scenarios being run without and with using keywords, respectively. The distribution charts indicate that the importance values calculated by the decision mechanism are mostly in the range of 15 to 40. This could result in uneven sensitivity of the decision

mechanism with respect to *user status*. In our definition of the *user status* variable we partially addressed this issue by scaling this variable in such a way that its value increases faster for the upper end than for the middle. This scaling can be optimized further towards the linearity of the decision mechanism characteristic.

In the next step we investigated which parameters used for calculating the importance value of the agent are effecting the importance values. Figure 9 shows which combinations of parameters were present in the simulated data and contributed in calculating the importance value of the agents with the test case scenarios. The following legend was used in the figure for particular parameters: P - agent priority, R - priority of the results, T - time limit, K - keywords found in the results. Only six of the possible sixteen combinations of parameters were used for two reasons. First, the priority criteria factor (P) is always present, eliminating the eight combinations that do not include P. Second, all simulated agents which allow the user to specify a time limit also allow the agent to prioritize its results. This eliminates the combinations which involve T without R (i.e. PT and PTK). This chart indicates that the combinations of parameters were essentially the same throughout all test scenarios. The leftmost bar is the average of the combinations contributing to the importance values in all three test scenarios.

Figure 10 shows the percentage of the parameters which were *critical* to the decision making process for different *user status* levels. Critical parameters were defined as the parameters that were necessary for the calculated importance value of the agent to be equal to or higher than the *user status* threshold. From Figure 10, it can be seen that as the *user status* threshold value decreases, the priority of the agent becomes the predominant critical criteria parameter. The other parameters have less significance in determining whether the agent should be filtered or not. At the

busy user status level, the only critical parameter is the priority of the agent. Even at the *flexible* level, priority is the predominant parameter. This concludes that the decision mechanism relies heavily on the priority of the agent.

8. Conclusions and Future Directions

As more software agents are being developed in the research community as well as for commercial products, the issue of agents competing for the user's attention and agent management should be recognized. What needs to be addressed is the issue of the user being overwhelmed by the number of agents processing tasks on a single desktop computer. This paper presents a possible solution to managing these agents for the user.

The goal of this research was to investigate the concept of a Desktop Agent Manager (DAM). Towards this goal, we implemented a prototype of a DAM and a simulation environment to generate various agents and to evaluate the performance of the DAM. Overall, the Desktop Agent Manager development achieved many of its goals. The Desktop Agent Manager captured, filtered, and managed the different types of agents; collected and managed requirements from the user; and displayed the results of the agents in a concise manner.

Our primary focus was on the decision mechanism of the DAM which makes the decision to either allow the agent to access the user or to filter the agent. Our goal was to develop a decision mechanism which can be understandable to the user and through which the user can easily control the number of agents not filtered by varying the *user status* variable and by using keywords. As

shown in the previous analysis, the Desktop Agent Manager performed as expected. As the *user status* level decreased, the number of agents not filtered increased proportionally.

All testing was performed both without using keywords and using a certain keyword set. We found that although keywords increased the importance values, the increase was not constant across all agents, since it depends on the keyword set defined by the user and the number of matches to the agent results. Since no one keyword set can be considered the "typical" keyword set chosen by any user, the only way to truly judge whether keywords have a major impact is to perform a user evaluation of the system with many users and keyword sets.

The analysis of the decision mechanism that we performed resulted in several suggestions for the design of this mechanism. We concluded that, depending on the values of *user status*, the sensitivity of the decision mechanism can become biased for or against some of the parameters. For instance, when the threshold value is low, (i.e., *bored* or *flexible*), the priority of the agent is the only parameter which is critical. When setting *user status* to *bored*, the user should understand that all agents will not be filtered because the priority of the agent is the only parameter taken into account. Unless we provide the user with the feedback showing them why the decisions were made, the user may never understand the reasoning behind the decision mechanism.

We showed that the characteristic of the agent can be optimized by the user towards a linear characteristic through adjusting the scaling of the *user status* parameter. Another possibility is to allow the user to change the weights for particular parameters. The next level of flexibility would be to allow the user to change the decision mechanism algorithm. This is a much more involved solution and it would require to provide the user with a programming language in which to set the

decision mechanism rules. The next step would be to incorporate machine learning into the Desktop Agent Manager. The Desktop Agent Manager would learn by watching how the user interacts with the agents that exist and generate its own rules.

While all of these solutions are feasible from the technical point of view, the final recommendation requires more research involving users. Any change in the decision mechanism leads to a learning curve for the user. For instance, in both the rule based and in the learning approaches, the DAM would have to have a language for defining new rules. The user would have to have a good understanding of this language in order to be able to either change or learn the new rules. Also, while a DAM with learning capabilities is desirable, the user of such an agent must be provided with a mechanism to control such an agent. The main decision now would be when to allow the agent to make independent decisions. To further analyze the effectiveness of the Desktop Agent Manager, a user study should be performed in which all of these possibilities should be evaluated.

References

Palaniappan, M., Yankelovich, N., Fitzmaurice, G., Loomis, A., Haan, B., Coombs, J., & Meyrowitz, N. (1992). The Envoy Framework: An Open Architecture for Agents. *ACM Transactions on Information Systems, Volume 10, Number 3*, 233-264.

Tessler, F. (1993). Magnet 1.0 intelligent agent-based utility. *Macworld, vol. 10, no.2*, 220(1).

Streeter, A. (1993). Open Sesame! summons agents to automate tasks. *MacWEEK, vol. 7, no. 21,* 1-2.

Adler, M., Durfee, E., Huhns, M., Punch, W., & Simoudis, E. (1992). AAAI Workshop on Cooperation Among Heterogeneous Intelligent Agents. *AI Magazine, Summer*, 39-42.

Funke, D., Neal, J., & Paul, R. (1993). An approach to intelligent automated window management. *International Journal of Man-Machine Studies, Vol. 38,* 949-983.

Avouris, N., Van Liedekerke, M., Lekkas, G., & Hall, L. (1993). User interface design for cooperating agents in industrial process supervision and control applications. *International Journal of Man-Machine Studies*, Vol. 38, 873-890.

Bird, S. (1993). Toward a taxonomy of multi-agent systems. *International Journal of Man-Machine Studies, Vol. 39*, 689-704.

Zlotkin, G. & Rosenschein, J. (1993). A Domain Theory for Task Oriented Negotiation. *Proceedings of IJCAI'93*, 416-422.

Maes, P. & Kozierok, R. (1993). Learning Interface Agents. *Proceedings of AAAI '93,* 459-465.

Kozierok, R. and Maes, P. (1993). A Learning Interface Agent for Scheduling Meetings. *ACM SIG-CHI International Workshop on Intelligent User Interfaces*.

Sheth, B. & Maes, P. (1993). Evolving Agents For Personalized Information Filtering. *Proceedings of the 1993 IEEE Conference on AI Applications*.

Miley, M. (1993). Agent Technology: the fine line between smart design and intelligent software. *MacWEEK, vol. 7, no. 16*, 41-45.

Ercolini, D. A. G. (1994). *An Architecture for a Desktop Agent Manager*. Masters Thesis, Computer Systems Engineering/Engineering Software Design, Northeastern University, Boston, MA.

*importance := agent priority * 10;*

for each *keyword* in *keywords*

if *keyword* found in *message*

importance := importance + keyword weight;

if *keyword* found in *brief report*

*importance := importance + (keyword weight * 1.5);*

end for

where *keyword weight := keyword priority * keyword association level;*

importance := importance

*+ priority of the result * 2.5*

*+ (time // 15) * overdue time * 2.5*

where *overdue time :=* { *+1* if *(time // 15) <= time limit*

-1 if *(time // 15) > time limit*

if *importance < user status*

action := filter agent

else

action := do not filter agent;

Figure 1. DAM Decision Mechanism

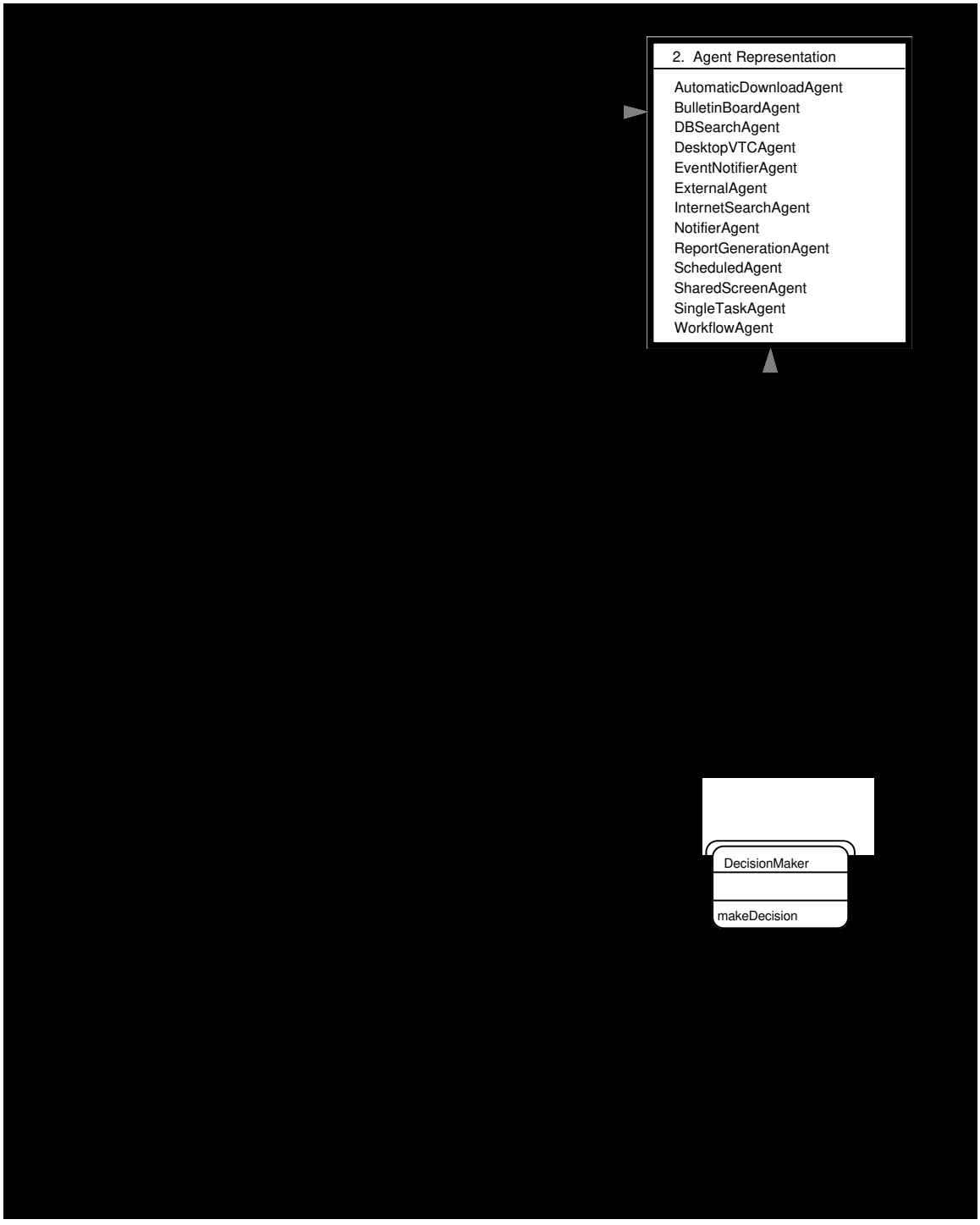

Figure 2. Components of the DAM Architecture

Name	Today's Application	Category	Description
Desktop VTC Agent	ShareVision	External	Provides live video teleconferencing on the desktop. Randomly invoked by external user.
Shared Screen Agent	Farallon Timbuktu	External	Allows the sharing of windows on the desktop.
DB Search Agent	SQL Sever, Oracle	Single Task	Query for information, Ad-hoc query, searching on specific keywords
Internet Search Agent	Gopher, WWW, Mosaic	Single Task	Search for information on the Internet
Bulletin Board Agent	Patty Maes' agents	Monitoring	Monitor for articles to be published on a bulletin board on the Internet
Automatic Download Agent	Lotus Notes	Scheduled	Scheduled Query for information. Notes automatically calls another Notes server and downloads any additions, deletions or changes to the replicated database
Report Generation Agent	Macros within Microsoft Word and Excel	Scheduled	Automatic report generator. Word/Excel can have macros which will generate a periodic report given a certain set of information exists.
Workflow Notifier Agent	Reach Workman	Monitoring	Sign this PR with 24 hours, review this document by next Tuesday
Event Notifier Agent	IBM's IntelliAgent	Monitoring	Budget is overrunning, schedule is slipping, someone changed a file

Table 1. Simulated Agents

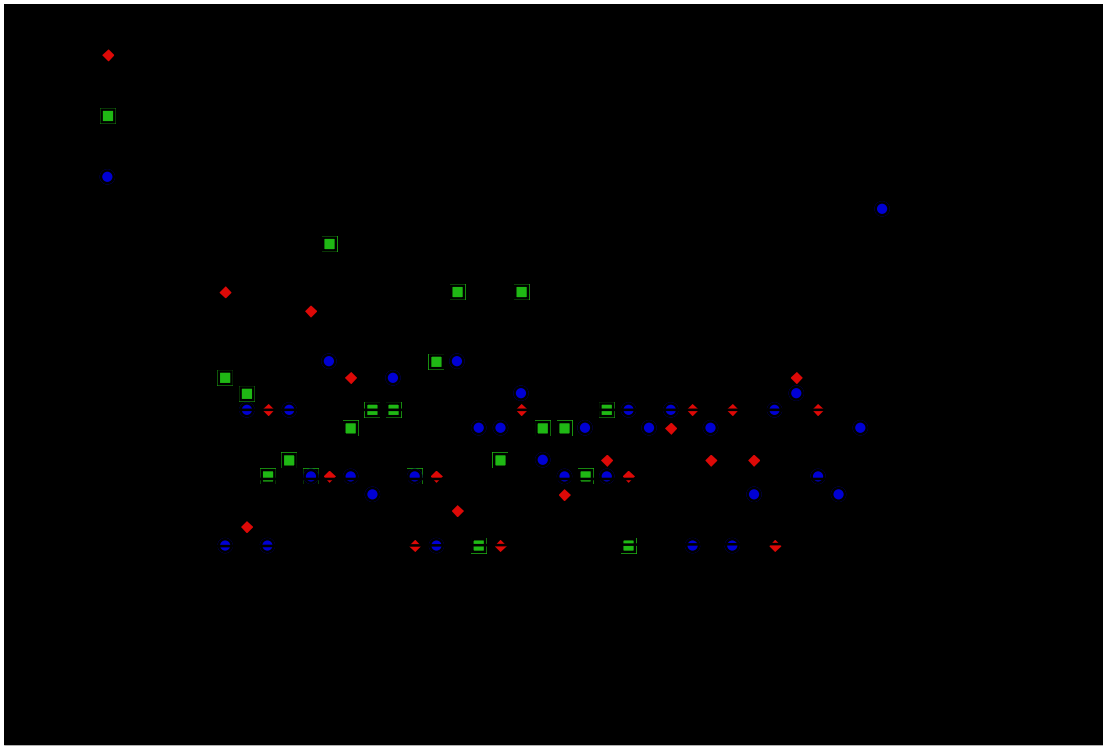

Figure 3. User Status Threshold Values for All Agent Sets

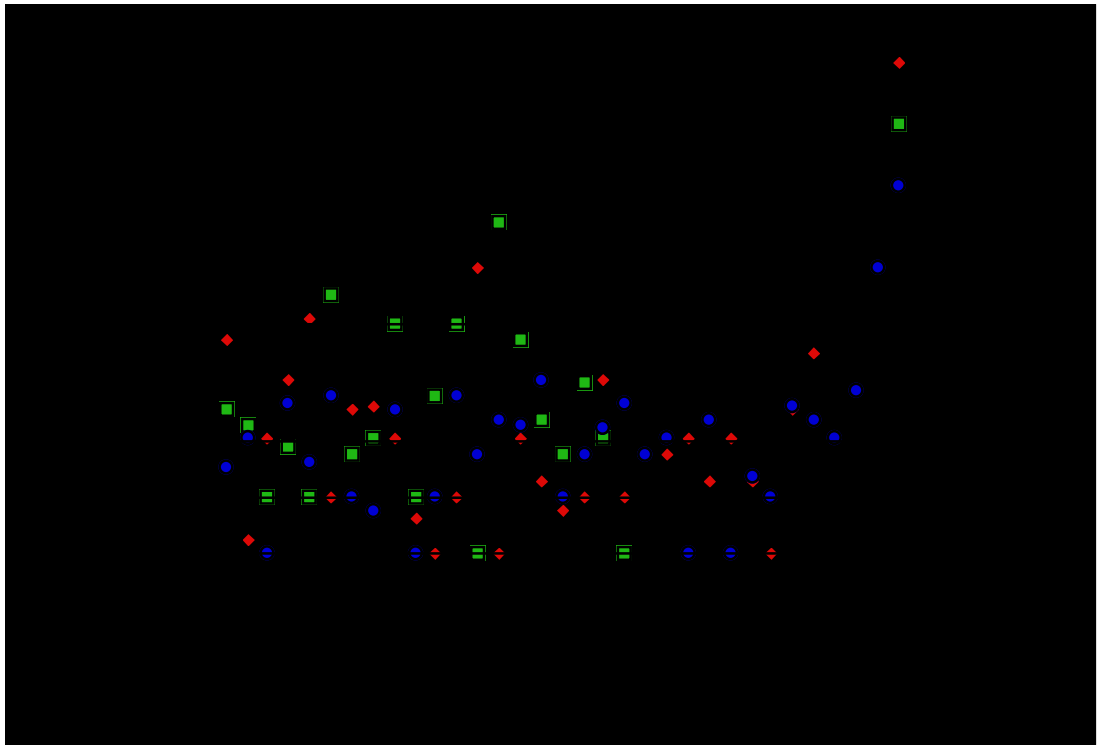

Figure 4. User Status Threshold Values for All Agent Sets Using Keywords

Figure 5. Average Percentage of Agents Not Filtered

Figure 6. Average Percentage of Agents Not Filtered Using Keywords

Figure 7. Average Distribution of Agent Importance Values

Figure 8. Average Distribution of Agent Importance Values Using Keywords

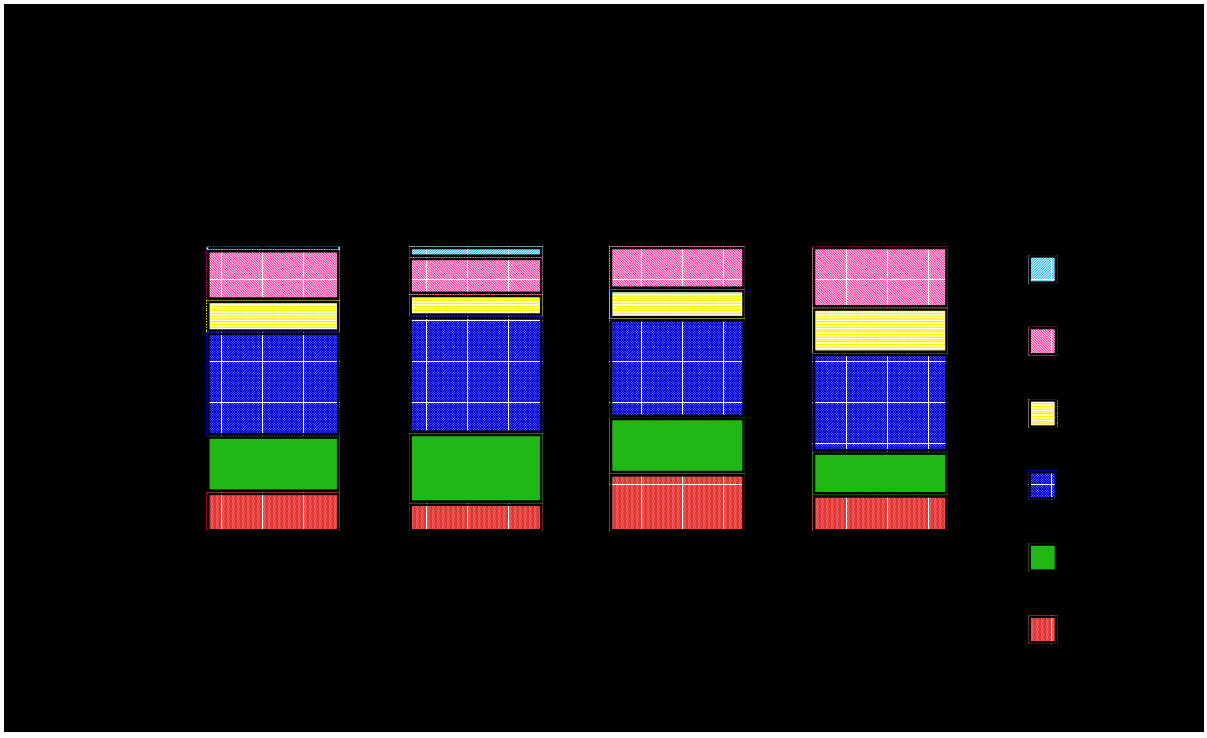

Figure 9. Percentage of Parameters Used by the Decision Mechanism

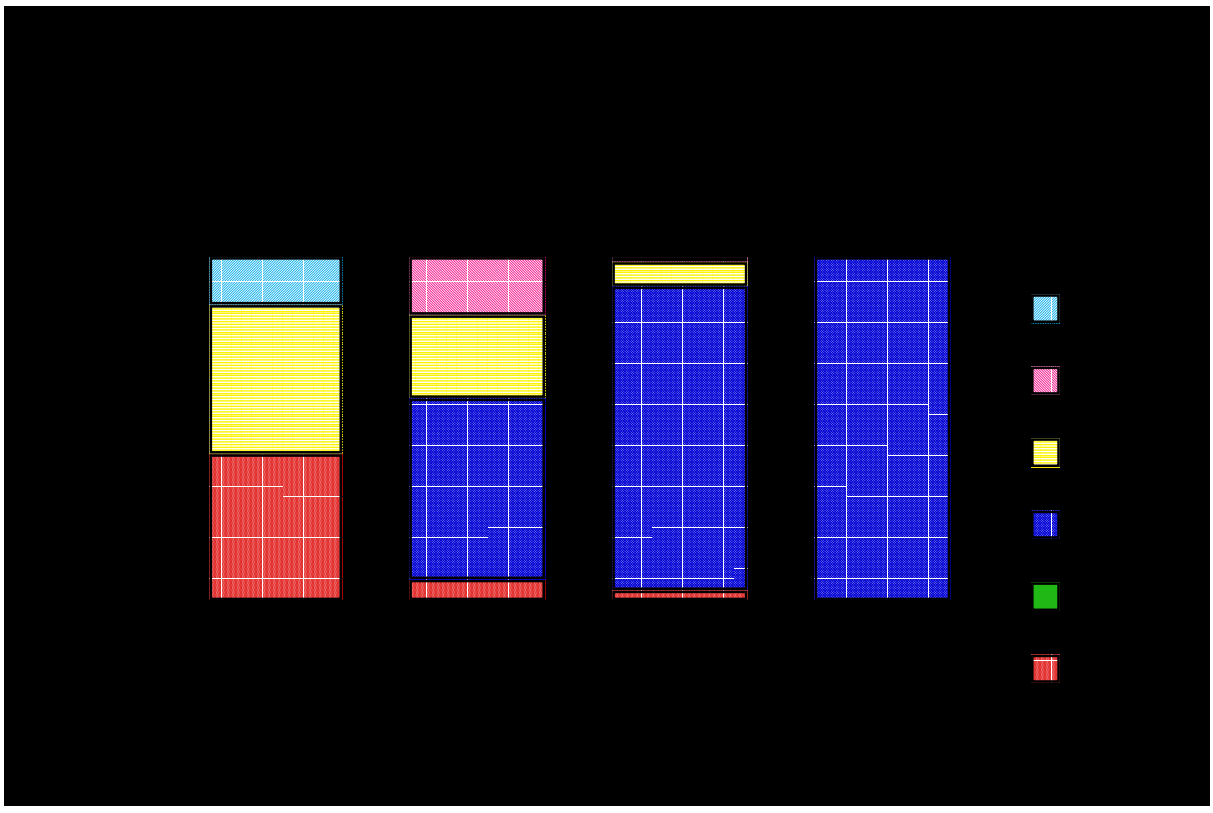

Figure 10. Percentage of Parameters Found to be Critical