A New Software Agent 'Learning' Algorithm

Pietro Murano

INTRODUCTION

This paper describes a new software agent (Bradshaw 1997) 'learning' algorithm. The philosophy this new algorithm is based on will also be outlined in the paper. The new algorithm 'observes' a user and, based on the user's action, or lack of actions, will make a set of 'inference(s)', and offer the user help/advice appropriate to the user's current situation. Furthermore, the algorithm can take into account a user's previous knowledge if this is made apparent by a user, while interacting with the system. The algorithm can be tailored in such a way that it can be of use to an agent in various virtual environments. The described 'learning' algorithm has been implemented with the aim of helping novices get started with UNIX commands (Gilly 1994). UNIX commands are typically very difficult for a novice to learn. The algorithm could be effectively modified to help novices learn other new interfaces, such as those found in command centres and control rooms. Such an algorithm would be a good tool in a training environment, as an addition to existing tools.

To that end this paper will consider:
- some of the existing related work on learning algorithms,
- a description of the new algorithm developed,
- areas where the algorithm could help in a command/control room environment.

SOFTWARE AGENT LEARNING ALGORITHMS

Some algorithms developed for software agents involve the agent 'observing' the user and making some kind of 'conclusion' on what the user has done. Then the agent will take some action based on this.

The software agent described in (Lieberman 1995) is such an agent. The aim of the agent is to 'observe' a user that browses the WWW, and based on the browsing pattern of the user, the agent will 'browse' in parallel relevant links. The algorithm incorporates heuristics and a best-first search. An innovative aspect of this software agent is the fact that a user does not need to communicate to the agent their intentions. Instead the agent attempts to discover what the user's intentions are by 'observation'.

In (Lieberman 1997), it is suggested that users at the interface give a system information 'for free'. It is advised that this information should be used by the agent to make inferences, with the aim of helping the user. Lieberman further suggests that an agent, particularly an autonomous one, can make use of a user's pause time at the interface.

In (Maes 1994), the idea of an agent 'learning' its behaviour is proposed. The aim is for a software agent to 'observe' a user and based on the observation(s), provide appropriate user assistance. Maes also allows the option of an agent receiving instructions from the 'human' or even to learn from other software agents.

A NEW 'LEARNING' ALGORITHM

The new algorithm discussed here, uses a similar philosophy to the types of algorithms discussed by Lieberman and Maes. This new algorithm uses heuristics and 'observes' a user's actions. Based on these, the software agent makes 'decisions' or infers how to best assist a user. Furthermore the user can communicate with the agent via Automatic Speech Recognition (ASR).

The setting used for the new algorithm was UNIX commands. For beginners this is a notoriously difficult task requiring effort and perseverance. Furthermore, beginners may be computing beginners in general or beginners to UNIX only. Someone could be an expert in DOS, but know nothing of UNIX commands. Even for experienced users it can be awkward to remember 'a particular' command for accomplishing a specialised purpose. The difference with this scenario is that experienced users will be in a better position to find the information they need.

These aspects were considered in the development of the prototype learning algorithm for the software agent. A further consideration was the wish to interfere as little as possible with the original UNIX X-Windows look and feel. This was so that a user did not need to learn 'another' interface, but would only need to interact with the software agent.

Assuming a user has begun a session at a console, the software agent immediately begins by observing the user. If the agent sees that the user has not hesitated, but has made a successful start, the software agent expects a valid command to be entered. If this is the case, then the software agent waits in the background continually observing the user's behaviour. This can be illustrated as shown in Figure 1:

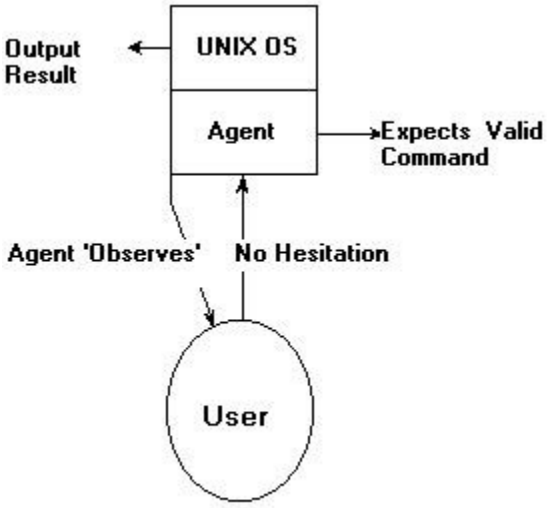

Figure 1: No user hesitation

If the next command to be entered by the user is non-existing, then the software agent offers assistance, but does not take control. This is because the agent infers that the situation is the result of a mistaken key stroke. However at the same time the agent assigns a weight in this situation, so that if mistaken key strokes should habitually continue, the agent will eventually take control based on the weighting. This is illustrated in Figure 2 and assumes the actions in Figure 1 have just taken place:

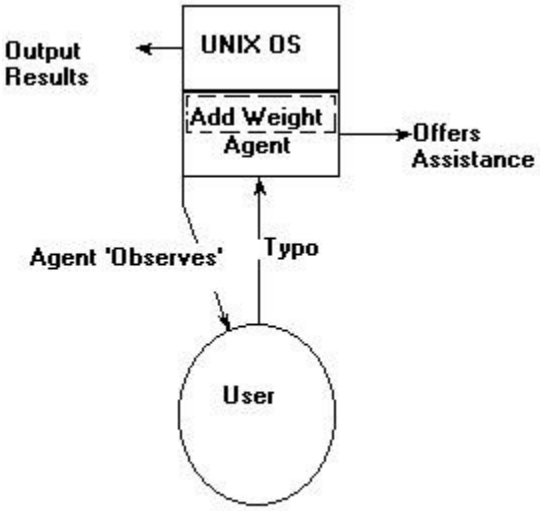

Figure 2: Non-existing command entered

If though another user should be a complete novice to UNIX and arrives at the interface and hesitates excessively, the agent will infer that the user is possibly a novice and take control. This is illustrated in Figure 3:

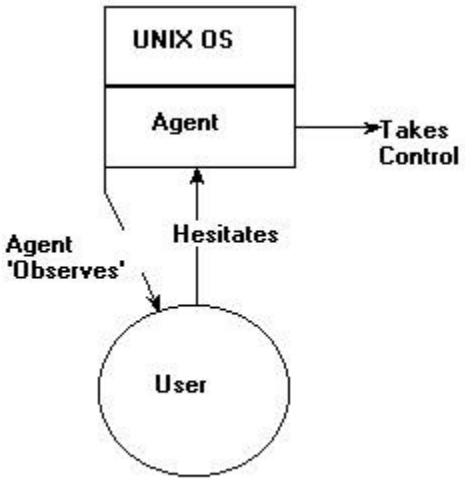

Figure 3: Novice UNIX user

The agent will then ask the user to input (via the microphone) what they intend to do. Upon receiving a request the agent will 'show' the user what to do, to achieve their intentions. This is illustrated in Figure 4:

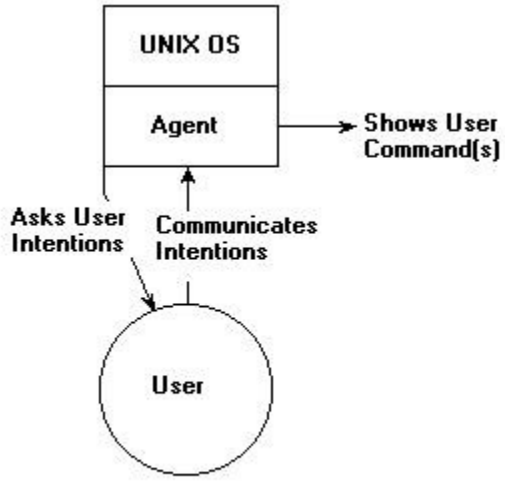

Figure 4: User asks agent for help

The agent will then wait to see if the user is following the instructions provided. If the user does not follow the instructions the agent will further attempt to help the user. A similar thing happens if the user asks the system for help. The agent infers that there is evidently a problem or that the user is a beginner. The agent also 'bows out gracefully' if upon asking the user what they want to do, the user does not reply but enters a correct command. The agent in this circumstance does not 'insist' on a reply, but returns to 'observing' in the background.

If a user happens to be knowledgeable in DOS and tries to use DOS, the agent detects this and offers a translation (if available) to the equivalent UNIX command. This is illustrated in Figure 5:

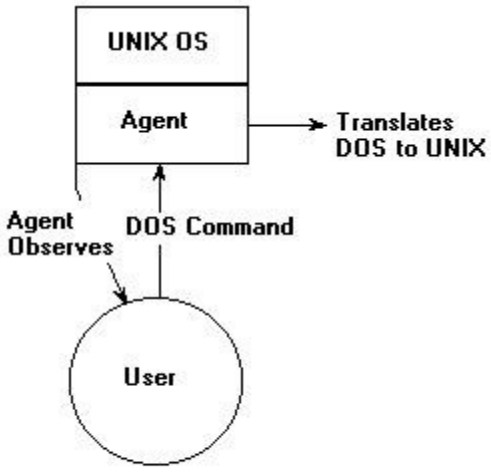

Figure 5: DOS user

If the user enters the appropriate command, the agent will continue to assist the user. If though the user does not return the expected command, the agent will attempt to discover what the user is wanting to achieve by asking the user. This behaviour also applies to typing errors that a user can do. These would be errors where it could be inferred what the user is trying to accomplish, e.g. if the user enters 'pw d', then the agent can infer that the user is

trying to input 'pwd' and advise that there should be no spaces in 'pwd'. This is a simple example, but the principle is equally applicable to more complex commands or groups of commands where it would be more difficult for a user to see their own mistake.

The author was eager to ensure that the help given was not annoying, especially as the user was typing. The algorithm in line with the philosophy discussed (see 'Software Agent Learning Algorithms above) 'observes' a user. This means that an inference is made after an observation. Then help is given. A contrast to this algorithm is the predictive text composition available in some mobile telephones, e.g. (Nokia 2000). This feature tried by the author, in a Nokia telephone, is found to be annoying and the prediction is found to be hardly ever correct. The idea is supposed to help a user to enter text (for text messaging) more quickly and conveniently. However in the author's experience the most efficient way to enter text is to switch off the predictive text input.

APPLICATION OF ALGORITHM TO OTHER DOMAINS

The idea of software agents being used in a control room environment for co-operative working is already being explored, e.g. (Patel, Mitchell et al. 1999). Hence, the principles and philosophy discussed above could be used in other domains. All that would be required are some modifications to ensure the algorithm is pertinent to the particular domain. The algorithm could be implemented to help novices learn new or non-standard interfaces. This would be suitable in command centres and control rooms. The algorithm could further be modified for use in an 'update' to an operator's training (discussed below).

The paper by (Boussoffara and Elzer 1999), discusses certain problems in a control room environment. This particular experiment was in the context of a coal fired power station simulation. Their experiment included different types of interfaces. These were based on Piping and Instrumentation (P&I) diagrams. They also used Mass Data Display (MDD) - involving a graphical representation comprising of the whole process and Trend Graphs - which display the history of a process variable. Combinations of these interfaces were tested together.

They concluded that 'pattern matching displays', e.g. MDD, were faster for detecting process anomalies, but seemed to 'cause' users to make judgements too early on. Instead 'time related' information, e.g. Trend Graphs, seemed to 'cause' less errors. They also noticed that subjects tended to use the MDD interface for categorisation and diagnosis. The problem being that the MDD interface was not designed for this purpose. They suggest developing new interfaces for resolving some of these problems (Boussoffara and Elzer 1999).

One of the ways to help this issue would be to have more than one type of interface available simultaneously, to be used collectively. Furthermore, an agent algorithm of the kind described above could be included in such interfaces with the aim of guiding the operators. As Boussoffara et al point out, sometimes operators make erroneous judgements. If this happens the operator might initiate the wrong action. The agent algorithm could infer this situation based on the operator's action(s), and on the 'knowledge' it would hold regarding the current process problem. It could then go on to provide assistance in a similar way to the UNIX navigation issue described above.

Another application the algorithm could have is to use it in a training environment, so that operators do not become used to making judgements too early on in the process of diagnosing a problem, i.e. it would help an operator to avoid developing bad habits.

A further situation where the agent algorithm discussed could prove beneficial, is in the situation of a 'plant' converting from a 'hard desk' to a 'soft desk'. This issue of transition is discussed in (Dicken 1999), and it is stated that the operators in this situation face 'a major cultural change'. They discuss various criteria that were considered when designing the 'soft desk'. One interesting area was the issue of system navigation. They state the fact that 'soft desks' allow for much more data to be accessed compared with 'hard desks'. Furthermore, the data is not displayed all at once on a screen. They describe various standard Graphical User Interface (GUI) techniques, such as hotspots and buttons. The descriptions also include using function keys.

Dicken acknowledges that operators require training for using the 'soft desk', and that after an initial period operators become proficient. Learning to use a new interface whether it is GUI based, command based or non standard, requires dedication and effort. The agent learning algorithm could be implemented to ease this initial period of learning. A training room could be set up where operators in their training cover various scenarios. As they attempt these with the 'soft desk', the agent algorithm could guide the navigation to the appropriate screen/hotspot/virtual control. ASR could be used, where operators could request the agent for information. Hence, if an operator cannot remember how to access a particular control, the agent could make certain inferences and provide appropriate help. Then the results of this initial operator query could be used by the agent to further help the operator with other issues, e.g. inferring that it may be difficult for this particular operator to find certain controls.

CONCLUSIONS

The described algorithm has been developer tested in the UNIX domain, for a sub-set of the UNIX commands. The algorithm works well in the described situations. Particularly useful is the feature of correcting spelling mistakes and/or syntax errors for known commands, e.g. 'ls -l' can be written by novices as 'ls-l'. The software agent has also been developed with the option of being switched off. Some agents can become annoying when it's awkward to turn them off, e.g. Microsoft 'paper clip'.

It has also been partially tested with users who were non UNIX users. The algorithm is proving itself to be of help to users who would not normally be able to navigate their way around the UNIX interface on their own. The algorithm brings the results of (with little training) a new user being able to at least get started in using the UNIX commands. Users find the spelling/syntax help that the agent gives to be useful, because errors such as those mentioned in the previous paragraph are common in new users. The agent is also good for prompting a user if they hesitate excessively.

A negative observation, made during the developer testing and also with novice users, is that the agent is more difficult to predict in its actions, because it is trying to make accurate inferences based on the user's actions or lack of them. This means that the help given by the agent might be different a second time around in an interaction. This is a concern that Ben Shneiderman in (Bradshaw 1997), uses as a reason for his scepticism towards software agents.

It would be interesting and useful for the algorithm described to be implemented in the situations described above from the cited research papers. Such an implementation could help in training and every day work within a control room or command centre.

REFERENCES

Boussoffara, B. and P. F. Elzer (1999). Evaluation of Interfaces for S&C of Large Technical Systems. People in Control An International Conference on Human Interfaces in Control Rooms, Cockpits and Command Centres, University of Bath, UK, IEE.

Bradshaw, J. M. (1997). Software Agents, AAAI Press, MIT Press.

Dicken, C. R. (1999). From Hard Desk to Soft Desk - The Changing Operator Interface. People in Control An International Conference on Human Interfaces in Control Rooms, Cockpits and Command Centres, University of Bath, UK, IEE.

Gilly, D. (1994). UNIX In A Nutshell, O'Reilly and Associates.

Lieberman, H. (1995). Letizia : An Agent That Assists Web Browsing. International Joint Conference on Artificial Intelligence, Montreal.

Lieberman, H. (1997). Autonomous Interface Agents. ACM Conference on Human-Computer Interface [CHI-97], Atlanta, USA.

Maes, P. (1994). "Agents That Reduce Work and Information Overload." Communications of the ACM 37(7): 31-40,146.

Nokia (2000). Nokia 3210. http://www.nokia.com/phones/3210/index.html.

Patel, R., R. J. Mitchell, et al. (1999). Co-operative Task Support For the Control Room. People in Control An International Conference on Human Interfaces in Control Rooms, Cockpits and Command Centres, University of Bath, UK, IEE.