Contemporary Engineering Sciences, Vol. 8, 2015, no. 18, 841 - 851
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2015.55161

Intelligent Agents for Automated Management of User Files

Kirill Mishchenko

Department of Algebra and Discrete Mathematics
Ural Federal University
620083 Ekaterinburg, Russia

Abstract

There is high popularity of research related with designing intelligent agents. But the issue of designing intelligent file agents is investigated poorly. In this paper we investigate different aspects of creating intelligent file agents. In particular, we implement a file system event observer. We also design a prototype of intelligent file agent using probabilistic neural networks. The implemented file system event observer does not make a significant load on a computer. Experiments with the prototype of intelligent file agent show high performance on modeled data. The prototype can be used to compare it with more advanced intelligent file agents.

Subject Classification: 68T42

Keywords: intelligent file agent, automated file management, file system event observer, probabilistic neural network (PNN)

1 Introduction

The problem of creating intelligent agents is investigated a lot in research [1]. In particular, we can mention research in application of intelligent agents in healthcare (see e.g. [2]), in education (see e.g. [3, 4, 5, 6]), in robotics (see e.g. [7, 8, 9]) and in human-computer interaction (see e.g. [10, 11, 12]). Among intelligent agents we can distinguish intelligent file agents. We will consider file agents as software agents which operate in a file system environment. As there are a lot of problems which can be defined by using the terms files and operations on them, there are different possible applications of file agents. In particular, in [13] software agents helping to maintain a UNIX file system for better disk space utilization by compressing and backing up are presented. There is an agent organizing Start menu to get an access to the most frequently used programs in the operating system Microsoft Windows XP [14]. Another application of file agents consists in finding a data placement on heterogeneous storage devices to optimize the data access performance for a particular user [15]. In this paper we consider the following issue. Computer users need to pay attention to keep their files well-organized. For example, users take into consideration what actions to apply to downloaded files. Possible options are deletion and moving to a particular directory. Thus, there is the issue of creating the intelligent file agent, which will be able to automate management of user files by offering the likeliest actions with them. In spite of that there is high popularity of research related with designing intelligent agents, the issue of designing intelligent file agents is investigated poorly. As there are practical applications of file agents, research in this area is considered to be reasonable. This paper provides an architecture of file agents, consideration of observing a file system and research on the design of a simplified model of intelligent file agent. The results can be used in future for comparing with more complicated models.

2 File agents architecture

There are several approaches to define an agents architecture. Let consider the abstract architecture for agents with state described in [16]. Let S be a set of all possible states of the environment, which an agent occupy; A be a set of all possible action of the agent; I be a set of all possible states of the agent; P be a set of all possible percepts. Than, how the agent operates can be represented through the functions

$$see : S \to P, next : I \times P \to I, action : I \to A.$$

The function *see* is used to get information about the environment. The function *next* is used to define the next state of the agent. The function *action* is used to make the decision about applying action to the environment. It is assumed, that the agent starts with the state i_0. The agent operates in a loop by observing environment, changing it's state, making a decision to do some action and performing it. Let apply this architecture to file agents. Let the state $s \in S$ contains a file tree structure, information about files and their content. We want to learn to generate file actions corresponding to the user's intentions. One of approaches to build complex intelligent systems is to delegate different functions to subsystems [17]. We can try to apply it to file agent designing. The first subsystem will observe file system events. The next subsystem will extract knowledge from observed information. The last subsystem will apply the extracted knowledge to help the user to manage files. This delegation can be used to represent file agents as agents having the vertically layered architecture [18] illustrated in figure 1.

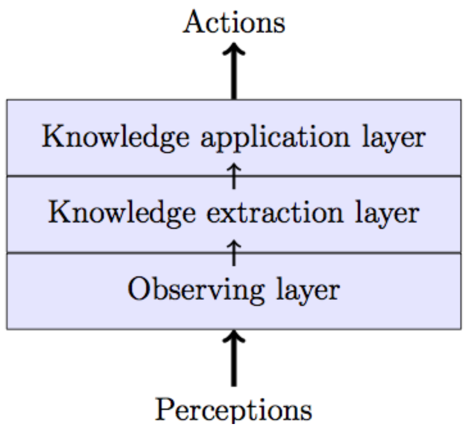


Figure 1: Vertically layered architecture of file agents

Information about moving files can be useful when we teach agent about how users manage their files. In section 3 we consider the problem of designing the observer logging information about creating, moving and deleting files. The generating actions strategy of file agents can be based on the history of the user's actions. In this case it is reasonable to generate the same actions for similar files. To define the degree of similarity it is possible to use different kinds of information about files, for example size, creation time, extension, tags, name. To process file names it is required to carry out a large-scale complex experiment. As a first approximation, it is useful to consider some simplified model of intelligent file agents. In section 4 we examine file agents processing a limited amount of information about files.

3 File system monitoring

We develop our observer in Mac OS X. There are several challenges during development the observer. The first one is the following. There is the opportunity to get the type of an observed file system event in Mac OS X by using the system API [19]. However, it is not enough to get full information about the event. It takes place because there is no file identifier passed when the event happens, just the file name. When the user moves a file there are two system events. For the first one the old file name is passed. For the second one the new file name is passed. Without the file identifier we cannot say that both system events refer to the same file. Thus, we cannot use event types passed by the operating system. There is another challenge. Before we have started to process a new system event other events could happen with the related file. It should be taken into consideration. There is yet another challenge. In Mac OS X when some file is downloaded from Internet through the standard browser Safari, at the beginning the browser creates the temporary file storing downloaded content. After the file is completely downloaded, the temporary file is renamed to the original name. In other words, there is a moving event. When downloaded files are small and downloading processes are mostly instantaneous, two different file system events - creating and moving - can fail to be handled in time. In our system we want to log all file system events occurring after downloading files as creating events. To overcome the mentioned issues we can do the following: storing a file system snapshot with file identifiers and using it to define the type of a file system event and the related file identifier; assuming that other file system events could happen before we start to process the event; adding filters that convert moving events for downloaded files into creating events. We discuss the system in more details in the section 3.1.

3.1 The structure of the observer

We create the observer using object-oriented design. It is written in C++. The observer contains following classes: 1) the class providing processing events; 2) the class which provides operations for creating, accessing and modifying file system snapshots; 3) the classes providing filters of events; 4) the class logging file system events to xml file. We will assume that a file system snapshot is a file system tree containing names and identifiers of files. In the case of Unix-style file systems, identifiers of files are inodes. At the beginning a file system snapshot is created with an instance of the class 2. When a new file system event occurs, an appropriate function is called. The input parameter of the function is the file name corresponding to the event. The file name is passed to an instance of the class 1. The instance of the class 1 interacts with the instance of the class 2. It tries to get the file identifier from the maintained snapshot using the class 2. It also tries to get the file identifier by using the appropriate system function. Depending on successful of both tries and comparing the gotten identifiers, we make the conclusion about which file system event has happened. Possible cases are creating, moving and deleting the file. Further, information about the event passes to an instance of the class 4. The filter classes 3 can be used to drop some events or to correct information in them.

One of the most complex parts of the system is the class maintaining file system snapshots. Snapshots are stored with the following structure:

```
struct FileNode
{
    std::string fileName;
    ino_t inode;
    std::map<std::string, FileNode*> *children;
    FileNode *parent;
};
```

The field *fileName* contains a file name (not a full path). The field *inode* contains the identifier of the file. The field *children* contains the pointer to map from names of child files to internal structures. It is NULL if there are no child elements. The field *parent* contains the pointer to the structure of the parent file (folder). It is NULL if there is no parent folder.

The filter classes allows to narrow the range of observed file events. It is useful if we want to log just events related with a particular folder such as Downloads.

3.2 Testing

While the observer is running, it is logging file system events to a XML file. Each entry contains the type of an event; the path of the file with which the event is related; the timestamp of the event. For moving events both old and new paths are logged. There are following types of events: creating a file has the type 1; removing a file has the type 2; moving a file has the type 3. Example of logged events are shown on the figure 2.

```
<FileSystemEvent timestamp="1395300764791365" eventType="1" path="/Users/kirill/Downloads/
    document.pdf"/>
<FileSystemEvent timestamp="1395300788820560" eventType="3" path="/Users/kirill/Downloads/
    document.pdf">
   <MovedEvent newPath="/Users/kirill/Documents/document.pdf"/>
</FileSystemEvent>
<FileSystemEvent timestamp="1395301600617726" eventType="1" path="/Users/kirill/Downloads/
    document2.pdf"/>
<FileSystemEvent timestamp="1395301936831667" eventType="3" path="/Users/kirill/Downloads/
    document2.pdf">
   <MovedEvent newPath="/Users/kirill/.Trash/document2.pdf"/>
</FileSystemEvent>
```

Figure 2: Examples of logged events

It should be noted that the implemented observer does not make a significant load on the tested computer. The load is inconsequential either in the case of lack of file system events or in the presence of them. More precisely, the usage of CPU for both cases was less than 0.1%.

4 A prototype of intelligent agent

In this section we will consider modeling an activity of the user which downloads files from the Internet and chooses actions for them. Further we will design the intelligent agent which is able to manage files of such user automatically.

4.1 Modeling a user activity

Let each file have n features (for example, creation time, size, etc.). Let $x_1 \in X_2, x_2 \in X_2, ..., x_n \in X_n$ be these features. For brevity, the tuple $(x_1, x_2, ..., x_n)$ will be called the file. We use $\overline{x}$ and $\overline{X}$ to denote the tuple $(x_1, x_2, ..., x_n)$ and $X_1 \times X_2 \times ... \times X_n$ accordingly.

To model downloading files from Internet by a user and doing actions with them, we will need the following things. The file generator to produce $\overline{x} \in \overline{X}$. The function $f(\overline{x}, y) : \overline{X} \times \{0,1\} \to R$, which defines the desire of the user to download $(y = 1)$ or to not download $(y = 0)$ the file $\overline{x}$ from the Internet. The $g(\overline{x}, a) : \overline{X} \times N \to R$, which defines the desire of the user to do the action a with the file $\overline{x}$. If there are M_{act} possible actions, then $a \in \{1, 2, ..., M_{act}\}$.

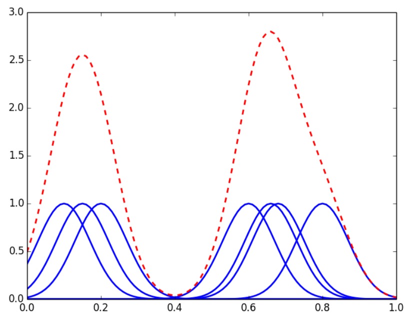
We model the user activity with iterations. Let the user download K files per iteration. On i-th iteration the user downloads the set of files F_i which is defined as $F_i = \{\overline{x_i^1}, ..., \overline{x_i^K}\}$, $\overline{x_i^k} \in \overline{X}$, $(\overline{x_i^k}, 1) > f(\overline{x_i^k}, 0)$, $k = 1..K$. The inequalities $f(\overline{x_i^k}, 1) > f(\overline{x_i^k}, 0)$, $k = 1..K$, can be interpreted as the desire to download the files $\overline{x_i^1}, \overline{x_i^2}, ..., \overline{x_i^K}$ over the desire to not download them. For following discussing we need define $G_i = \{(\overline{x_i^1}, a_i^1), ..., (\overline{x_i^K}, a_i^K)\}$, $\overline{x_i^k} \in F_i$, $a_i^k = argmax_a g(\overline{x_i^k}, a)$, $k = 1..K$. a_i^k defines the most desired action for the file $\overline{x_i^k}$. For mathematical rigour, each of the functions $g(\overline{x_i^k}, a)$, $k = 1..K$, should have a single maximum among all actions a.

4.2 Designing the file agent

We want the file agent to achieve an appropriate similarity in proposed actions with actions chosen by the user. In our model it means corresponding with actions that are matched to files in the set G_i.

Let assume $x_1 \in R, x_2 \in R, ..., x_n \in R$. In other words, $\overline{x} \in R^n$. The file agent need split the space R^n into subspaces such that each subspace is related with one desired action. The file agent also need continuously define more exactly such splitting while the user choose actions for new files.

We assume that we need to apply the same actions for files which are similar in the sense of the Euclidean distance. Then we can build class distribution functions using a form of the Gaussian distribution [20]. Let there be M different classes which. We enumerate them as $1, 2, ..., M$. Let for each class


(a)

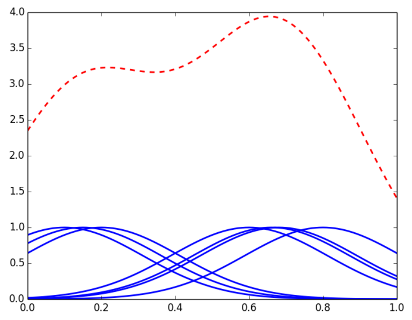

(b)

Figure 3: The class distribution for the training set $\{0.1, 0.15, 0.2, 0.6, 0.66, 0.68, 0.8\}$. The items of the sum (1) are drawn with solid lines. The result function is drawn with the dashed line. The parameter σ equals to 0.1 and 0.3 on (a) and (b) accordingly.

m we have the training set $\{\overline{x_m^1}, \overline{x_m^2}, ..., \overline{x_m^{K_m}}\}$. The class distribution function is defined as

$$h_m(\overline{x}) = \sum_{i=1}^{K_m} exp(\frac{-\|\overline{x} - \overline{x_m^i}\|^2}{\sigma^2}), m = 1..M. \quad (1)$$

We can find the likeliest class for $\overline{x}$ as $argmax_m h_m(\overline{x})$. This method matches how probabilistic neural networks work [21]. The parameter σ defines the width of the functions. For one-dimensional data it is illustrated with the figure 3. This method can be applied to design the file agent. In this case the classes are actions with files.

4.3 Testing

In our experiments file features x_1, x_2, x_3 are creation timestamp, size and file extension accordingly. Their ranges X_1, X_2, X_3 are specified at the beginning of the experiments and are used by the file generator.

For σ in (1) we use the value 0.1. In the end of each iteration we calculate the percentage of matching between the actions offered by the file agent and the appropriate actions from the set G_i. After that we use the set G_i for teaching the file agent.

To get mean results we carry out a composite experiment. We use polynomials of degree 3 for the functions f and g. The count of possible actions M_{act} varies from 10 to 20. For each M_{act} we conduct 100 experiments. As a result, we carry out $11 \cdot 100 = 1100$ experiments at all. Each experiment contains 100 iterations. The result of each experiment is a sequence of percentages of matching mentioned above. We can sort results of all experiments by sums of

Table 1: The quantitative results of experiments when the functions f and g are polynomials of degree 3

Handled cases	Matching rate after the first iteration	Matching rate in the last iteration	Average speed of changing matching rate	Amount of monotonicity changes
The average values among 10 worst cases	40.5 %	67.5 %	0.27 % per iteration	52.6
The average values among 100 worst cases	62.1 %	86.7 %	0.25 % per iteration	49.2
The average values among all experiments	91.3 %	97.9 %	0.07 % per iteration	15.2

matching in all iterations. Then the first sorted results can be considered as the worst cases.

We define the functions f and g by using polynomials. The coefficients of the polynomials are defined by random generator from the set $\{-1, 0, 1\}$. The parameters are normalized to the range $[-1, 1]$ with following formulas:

$$x'_j = -1 + (x_j - x_j^{min})/(x_j^{max} - x_j^{min}) \cdot 2, j = 1..3, \tag{2}$$

$$y' = -1 + y * 2, a' = -1 + (a-1)/(M_{act} - 1) \cdot 2,$$

where x_j^{min}, x_j^{max} are such that $X_j = \{x_j^{min}, x_j^{min} + 1, ..., x_j^{max}\}, j = 1..3$. Thus, the functions f and g are defined as

$$f(\overline{x}, y) = f(x_1, x_2, x_3, y) = f(x_1(x'_1), x_2(x'_2), x_3(x'_3), y(y')) = f'(x'_1, x'_2, x'_3, y'),$$
$$g(\overline{x}, a) = g(x_1, x_2, x_3, a) = g(x_1(x'_1), x_2(x'_2), x_3(x'_3), a(a')) = g'(x'_1, x'_2, x'_3, a'),$$

where f' and g' are polynomials with 4 variables.

Matching rate after the first iteration and in the last iteration, average speed of changing matching rate and amount of changing monotonicity for 10 and 100 words cases and also among all experiments are given in the table 1. It is seen that in the beginning the rates grow significantly. After that the rates vary in the range. However, the rates grow in general while increasing the number of iteration. Consideration of greater number of worst cases makes the relation more smooth. We carry out analogous experiments for the case when the functions f and g are polynomials of degree 5. The quantitative results of

Table 2: The quantitative results of experiments when the functions f and g are polynomials of degree 5

Handled cases	Matching rate after the first iteration	Matching rate in the last iteration	Average speed of changing matching rate	Amount of monotonicity changes
The average values among 10 worst cases	51.5 %	77.5 %	0.26 % per iteration	53.1
The average values among 100 worst cases	56.5 %	85.5 %	0.29 % per iteration	50.0
The average values among all experiments	88.1 %	97.2 %	0.09 % per iteration	19.0

experiments are given in the table 2. The experiments have the same tendency and show high performance of the method.

5 Conclusion

The file system observer was presented. The implemented observer did not make a significant load on the tested computer. The load was inconsequential either in the case of lack of file system events or in the presence of them. We designed and tested the file agent processing a limited amount of information about files. The experiments showed high performance of the proposed method. However, it should be noted that used probabilistic neural networks can make a meaningful load on a computer in the case of big data [20]. This issue was examined in [22] [23] and the methods to overcome it were proposed. It can be useful to investigate the applicability of the proposed methods to our issue. It seem interesting to examine another ways of defining files similarity. Additional information contained in files and file names can be used.

References

[1] D. Hoa Khanh and A. Ghose, Supporting change impact analysis for intelligent agent systems, *Science of Computer Programming*, **78** (2013), 1728-1750. http://dx.doi.org/10.1016/j.scico.2013.04.008

[2] L. Cardoso, F. Marins, F. Portela, A. Abelha, and J. Machado, Healthcare interoperability through intelligent agent technology, *Procedia Technology*, **16** (2014), 1334-1341.
http://dx.doi.org/10.1016/j.protcy.2014.10.150

[3] R. Villarica and D. Richards, Intelligent and empathic agent to support student learning in virtual worlds, *Proceedings of the 2014 Conference on Interactive Entertainment*, 2014, 1-9. http://dx.doi.org/10.1145/2677758.2677761

[4] M. Yaghmaie and A. Bahreininejad, A context-aware adaptive learning system using agents, *Expert Systems with Applications*, **38** (2011), 3280-3286. http://dx.doi.org/10.1016/j.eswa.2010.08.113

[5] N. Pandey, S. Sahu, R. Tyagi, and A. Dwivedi, Learning algorithms for intelligent agents based e-learning system, *IEEE 3rd International Advance Computing Conference*, 2013, 1034-1039. http://dx.doi.org/10.1109/iadcc.2013.6514369

[6] D. Szafir and B. Mutlu, Pay attention!: Designing adaptive agents that monitor and improve user engagement, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, 11-20. http://dx.doi.org/10.1145/2207676.2207679

[7] S. Rosenthal, J. Biswas, and M. Veloso, An effective personal mobile robot agent through symbiotic human-robot interaction, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 2010, 915-922.

[8] A. Gorbenko and V. Popov, Multi-agent path planning, *Applied Mathematical Sciences*, **6** (2012), 6733-6737.

[9] A. Gorbenko, V. Popov, and A. Sheka, Robot self-awareness: Exploration of internal states, *Applied Mathematical Sciences*, **6** (2012), 675-688.

[10] M. Ivanović, M. Radovanović, Z. Budimac, D. Mitrović, V. Kurbalija, W. Dai, and W. Zhao, Emotional intelligence and agents: Survey and possible applications, *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics*, 2014, no. 52. http://dx.doi.org/10.1145/2611040.2611100

[11] H. Ming, Z. Haibin, Z. MengChu, and G. Arrabito, Optimizing operator-agent interaction in intelligent adaptive interface design: A conceptual framework, *IEEE Transactions on Systems, Man and Cybernetics, Part C*, **41** (2011), 161-178. http://dx.doi.org/10.1109/tsmcc.2010.2052041

[12] J. Hyunsu, S. Chang Hwan, K. Moon Jeong, and E. Young Ik, Agent-based intelligent middleware for user-centric services in ubiquitous computing environments, *Journal of Information Science and Engineering*, **27** (2011), 1729-1746.

[13] H. Song, S. Franklin, and A. Negatu, Sumpy: A fuzzy software agent, *Proceedings of the ISCA Conference on Intelligent Systems*, 1996, 124-129.

[14] Description of the Start Menu in Windows XP, http://support.microsoft.com/KB/279767

[15] L. Wan, Z. Lu, Q. Cao, F. Wang, S. Oral, and B. Settlemyer, SSD-optimized workload placement with adaptive learning and classification in HPC environments, *30th Symposium on Mass Storage Systems and Technologies*, 2014, 1-6. http://dx.doi.org/10.1109/msst.2014.6855552

[16] M. Wooldridge, *Intelligent Agents: The Key Concepts*, Multi-Agent Systems and Applications II, Springer Berlin Heidelberg, 2002, 3-43. http://dx.doi.org/10.1007/3-540-45982-0_1

[17] R. A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, **2** (1986), 14-23. http://dx.doi.org/10.1109/jra.1986.1087032

[18] J. P. Müller, M. Pischel, and M. Thiel, *Modeling Reactive Behaviour in Vertically Layered Agent Architectures*, Intelligent Agents, Springer Berlin Heidelberg, 1995, 261-276. http://dx.doi.org/10.1007/3-540-58855-8_17

[19] Fseventstreameventflags - fsevents reference, https://developer.apple.com/library/mac/documentation/Darwin/Reference/FSEvents_Ref/#//apple_ref/doc/constant_group/FSEventStreamEventFlags

[20] R. Callan, *The Essence of Neural Networks*, Prentice Hall Europe, 1999.

[21] D. Specht, Probabilistic neural networks, *Neural Networks*, **3** (1990), 109-118. http://dx.doi.org/10.1016/0893-6080(90)90049-q

[22] M. Berthold and J. Diamond, Constructive training of probabilistic neural networks, *Neurocomputing*, **19** (1998), 167-183. http://dx.doi.org/10.1016/s0925-2312(97)00063-5

[23] R. K. Y. Chang, C. K. Loo, and M. Rao, A global k-means approach for autonomous cluster initialization of probabilistic neural network, *Informatica*, **32** (2008), 219-225.

Received: June 7, 2015; Published: September 9, 2015