Creating User Profiles from a Command-Line Interface: A Statistical Approach

José Antonio Iglesias, Agapito Ledezma, and Araceli Sanchis

Universidad Carlos III de Madrid,
Avda. de la Universidad, 30, 28911 Leganés (Madrid), Spain
{jiglesia,ledezma,masm}@inf.uc3m.es

Abstract. Knowledge about computer users is very beneficial for assisting them, predicting their future actions or detecting masqueraders. In this paper, an approach for creating and recognizing automatically the behavior profile of a user from the commands (s)he types in a command-line interface, is presented.

Specifically, in this research, a computer user behavior is represented as a sequence of UNIX commands. This sequence is transformed into a distribution of relevant subsequences in order to find out a profile that defines its behavior. Then, statistical methods are used for recognizing a user from the commands (s)he types. The experiment results, using 2 different sources of UNIX command data, show that a system based on our approach can efficiently recognize a UNIX user. In addition, a comparison with a HMM-base method is done.

Because a user profile usually changes constantly, we also propose a method to keep up to date the created profiles using an *age*-based mechanism.

1 Introduction

Would it not be interesting to recognize a computer user and to know how (s)he will behave after (s)he types a few commands?

Recognizing the behavior of others in real-time is significant in different tasks, such as to predict their future behavior, to coordinate with them or to assist them. In order to act efficiently, humans usually try to recognize the behavior of others. New theories claim that a high percentage of the human brain capacity is used for predicting the future, including the behavior of other humans [1].

Specifically, computer user modeling is the process of learning about ordinary computer users by observing the way they use the computer. This process needs the creation of a *user profile* that contains information that characterizes the usage behavior of a computer user. Experience has shown that users themselves do not know how to articulate what they do, especially if they are very familiar with the tasks they perform. Computer users, like all of us, leave out activities that they do not even notice they are doing. Thus, only by observing users we

can model his/her behavior correctly [2]. However, the construction of effective computer user profiles is a difficult problem because of the following aspects: human behavior is usually erratic, and sometimes humans behave differently because of a change in their goals.

In recent years, significant work has been carried out for profiling computer users. In this research, an approach for profiling and recognizing *general* user behavior profiles is proposed. This approach is called ***ABCD*** (*Agent Behavior Classification based on Distributions of relevant subsequences of commands*) and can be applied for creating and recognizing any behavior represented by a sequence of commands (or events). *ABCD* creates a user profile as a distribution of relevant subsequences and then statistical methods are applied for recognizing a given sequence of commands.

However, for evaluating *ABCD*, the UNIX operating system environment is used. The creation of the UNIX user profiles from a sequence of UNIX commands should consider the sequentiality of the commands typed by the user and the temporal dependencies. In a human-computer interaction by commands, the sequentiality of these commands is essential for the result of the interaction. This aspect motivates the idea of automated sequence learning for computer user behavior classification; if we do not know the features that influence the behavior of a user, we can consider a sequence of past actions to incorporate some of the historical context of the user. This aspect is taken into account in the HMM-based methods, so we will compare *ABCD* with a method which uses HMMs for modeling users. Finally, once the user has been classified, relevant actions can be done, however this task is not addressed in this paper.

This paper is organized as follows: Section 2 provides a brief overview of the background and related work relevant to this research. Our approach (*ABCD*) is explained in detail in section 3. Section 4 describes the experimental setting and the experimental results obtained. Section 5 compares the obtained results with a very well know technique (HMMs). The proposal for making *ABCD* adaptative is detailed in Section 6. Finally, Section 7 contains concluding remarks.

2 Background and Related Work

Different methods have been used to find out relevant information in the computer user behavior in different computer areas:

Discovery of navigation patterns: Spiliopoulou and Faulstich [3] present the *Web Utilization Miner WUM*, a mining system for discovering interesting navigation patterns in a web site. *WUM* prepares the web log data for mining and the language *MINT* mining the aggregated data according to the directives of the human expert. This work is complementary to "Footprints" tool, which focuses on the visualization of frequently accessed patterns and on the identification of pattern types that may be of importance [4].

Web recommender systems: Macedo et al. [5] propose a system (*WebMemex*) that provides recommended information based on the captured history of navigation from a list of known users. *WebMemex* captures information such as IP addresses, user Ids and URL accessed for future analysis.

Web page filtering: Gody and Amandi [6] present a technique to generate readable user profiles that accurately capture interests by observing their behavior on the Web. The proposed technique is built on the *Web Document Conceptual Clustering* algorithm, with which profiles without an a priori knowledge of user interest categories can be acquired.

Computer security: Pepyne et al. [7] describe a method using queuing theory and logistic regression modeling methods for profiling computer users based on simple temporal aspects of their behavior. In a similar area (intrusion detection problem), Coull et al. [8] propose an algorithm that uses pair-wise sequence alignment to characterize similarity between sequences of commands. The algorithm produces an effective metric for distinguishing a legitimate user from a masquerader. Schonlau et al. [9] investigate a number of statistical approaches for detecting masqueraders.

Although there is lot of work that focuses on user profiling in a specific environment, it is not clear that they can be transferred to other environments. However, the approach proposed in this research (*ABCD*) can be used in any domain in which a user behavior can be represented as a sequence of commands or events. Therefore, as sequences are very relevant in human skill learning and reasoning [10], the problem of user profile classification is examined as a problem of sequence classification. According to this aspect, Horman and Kaminka [11] present a learner with unlabeled sequential data that discover meaningful patterns of sequential behavior from example streams. Lane and Brodley [12] present an approach based on the basis of instance-based learning (IBL) techniques, and several techniques for reducing data storage requirements of the user profile.

3 ABCD: Agent Behavior Classifier Based on Distributions of Relevant Subsequences of Commands

Although *ABCD* can be applied for creating and recognizing any behavior profile represented by a sequence of commands, this research is focused on creating computer user profiles from a command-line interface. Specifically, *ABCD* is detailed using the UNIX commands environment.

ABCD, as other behavior modeling methods [13], uses a library in which all the different user profiles recognized are stored. Then, a matching of the sequence to classify with the *Profile-Library* is done. Thus, *ABCD* is divided into two phases:

1. **Construction of the User Behavior Profiles:** In this phase, the sequences of commands typed by different UNIX users are analyzed and the corresponding profiles are created and stored in the *Profile-Library*. This process is detailed in Section 3.1.

2. **User Classification:** The goal of this phase is to classify a *new* sequence of commands typed by a user into one of the profiles created in the previous phase. Section 3.2 explains the proposed statistical classification method.

3.1 Construction of the User Behavior Profiles

In this phase, the first step is to extract the significant pieces of the sequence of commands that can represent a pattern of behavior. When a user types a command, it usually depends on the previous typed commands and it is related to the following commands. According to this aspect, and as it was used in [14], in order to get the most representative set of subsequences from the acquired sequence, the use of a ***trie* data structure** [15] is proposed. This structure is also proposed in [16] to learn a team behavior and in [17] to classify the behavior patterns of a *RoboCup* soccer simulation team.

The construction of a user profile from a single sequence of commands is done by a three steps process: 1. Segmentation of the sequence of commands, 2. Storage of the subsequences in a *trie*, and 3. Creation of the user profile. These steps are detailed in the following 3 subsections.

In order to clarify the process for creating a UNIX user profile, let us consider the following sequence as example: $\{ls \rightarrow date \rightarrow ls \rightarrow date \rightarrow cat\}$.

Segmentation of the sequence of commands: Firstly, the sequence is segmented in subsequence of equal length from the first to the last element. Thus, the sequence $A{=}A_1A_2...A_n$ (where n is the number of commands of the sequence) will be segmented in the subsequences described by $A_i...A_{i+length}$ $\forall$ $i,i{=}[1,n\text{-}length{+}1]$, where *length* is the size of the subsequences created and determines how many commands are considered as dependent. In the rest of the paper, we will use the term *subsequence length* to denote the value of this length.

In the proposed sample sequence ($\{ ls \rightarrow date \rightarrow ls \rightarrow date \rightarrow cat\}$), let 3 be the subsequence length, then it is obtained: $\{ls \rightarrow date \rightarrow ls\}$ and $\{date \rightarrow ls \rightarrow date\}$ and $\{ls \rightarrow date \rightarrow cat\}$.

Storage of the subsequences in a *trie*: The subsequences of commands are stored in a *trie* in a way that all possible subsequences are accessible and explicitly represented. In the proposed *trie*, a node represents a command, and its *children* represent the commands that follow it. Also, each node keeps track of the number of times a command has been inserted on to it. As the dependencies of the commands are relevant in the user profile, the subsequence suffixes (subsequences that extend to the end of the given sequence) are also inserted.

Considering the previous example, the first subsequence ($\{ls \rightarrow date \rightarrow ls\}$) is added as the first branch of the empty *trie* (Figure 1a). Each node is labeled with the number 1 (in square brackets) which indicates that the command has been inserted in the node once. Then, the suffixes of the subsequence ($\{date \rightarrow ls\}$ and $\{ls\}$) are also inserted (Figure 1b). Finally, after inserting the 3 subsequences and its corresponding suffixes, the completed *trie* is obtained (Figure 1c).

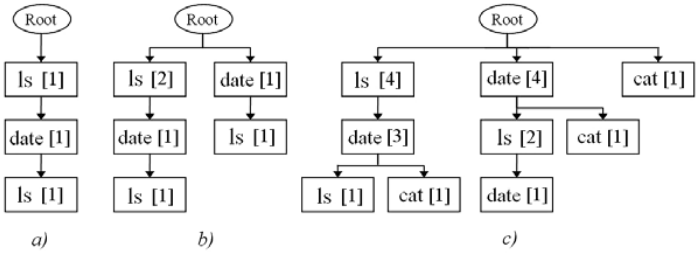

Fig. 1. Steps of creating an example trie

Creation of the user profile: For this purpose, **frequency-based methods** are used. Specifically, to evaluate the relevance of a subsequence using *ABCD*, its relative frequency or support [18] is calculated. In this case, the **support** of a subsequence is defined as the ratio of the number of times the subsequence has been inserted into the *trie* to the total number of subsequences of equal size inserted. Calculating this value, the *trie* is transformed into a set of subsequences labeled with its corresponding support value. This structure is represented as a distribution of relevant subsequences. Once a user behavior profile has been created, it is stored in the *Profile-Library* with an identification name.

In the previous example, the *trie* consists of 9 nodes; therefore, the profile consists of 9 different subsequences which are labeled with its support (Figure 2).

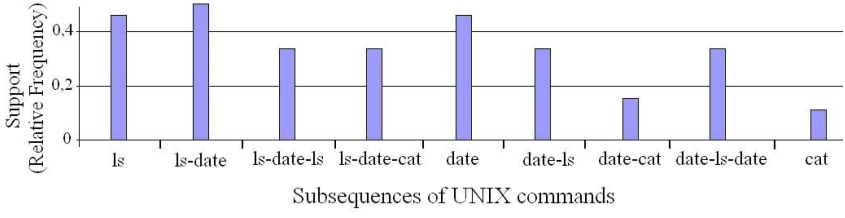

Fig. 2. Distribution of subsequences

3.2 User Recognition

In this second phase, a *new* sequence of commands typed by one of the users previously analyzed must be classified. It means that given an observed sequence E typed by a user and a set of user behavior profiles $P = \{up_1, up_2, ..., up_n\}$ stored in the *Profile-Library*, the goal of this phase is to determine into which profile $up_i \in P$ the sequence E belongs to.

Firstly, the distribution of relevant subsequences of the *new* sequence (input) is created by applying the process explained in the previous section. Then, it is matched with all the profiles stored in the *Profile-Library*. As both profiles are represented by a distribution of values, a statistical test is applied for matching

these distributions. A non-parametric test (or distribution-free) is used because this kind of tests does not assume a particular population distribution. The proposed test applied for matching two behaviors is a **modification of the *Chi-Square Test*** for two samples.

To apply the proposed test, the sequence to classify (input) is considered as an observed sample and the profiles stored in *Profile-Library* are considered as the expected samples. Then, this test compares the observed distribution with all the expected distributions objectively and evaluates whether a deviation appears.

The *Chi-Square Test* compares the two sets of support values in which *Chi-Square* is the sum of the terms $\frac{(Obs-Exp)^2}{Exp}$ calculated from the observed (Obs) and expected (Exp) distributions. However, using this test, all the expected values are compared but if an observed value is not represented in the expected distribution, it is not considered. Also, the number of subsequences in an expected distribution is usually very large, so this kind of comparison can be very time-consumed. In order to solve these *problems*, the way to compare the two distributions is modified to the sum of the terms $\frac{(Exp-Obs)^2}{Obs}$.

An important advantage of the proposed test is its rapidity because only the observed subsequences are evaluated. However, there is no penalty for the expected relevant subsequences which do not appear in the observed distribution.

Using this test, a value that indicates the deviation between the observed and the stored profile is obtained. This deviation needs to be calculated with all the profiles stored in *Profile-Library* and the profile that obtains the lowest deviation value indicates the closer similarity. Also, the number of terms to sum in each comparison is always the same: number of subsequences of the observed profile. It means that the degrees of freedom (dof) are the same in all the comparisons with the expected behavior profiles. Otherwise, a normalization of the results according to the dof should be done.

As example, let us consider that the sequence that represents the observed behavior is: $\{ls \rightarrow date \rightarrow cpp\}$. Figure 3 shows the comparison between the previous expected distribution (*Expected Profile 1*) and the observed distribution (*Observed Profile*). Obtaining the support value of each subsequence in Figure 3, the deviation value in this example is: $\frac{(0{,}44-0{,}33)^2}{0{,}33} + \frac{(0{,}5-0{,}5)^2}{0{,}5} + \frac{(0{,}44-0{,}33)^2}{0{,}33} + \frac{(0-1)^2}{1} + \frac{(0-0{,}5)^2}{0{,}5} + \frac{(0-0{,}33)^2}{0{,}33}$.

4 Experimental Setup and Results

For evaluating *ABCD* in the UNIX environment, we have used 2 different sources of UNIX data with different number of users to classify:

- **Set of 9 UNIX Users:** Data[1] drawn from the command histories of 9 UNIX computer users at Purdue University over 2 years [19]. Each user file contains from about 10000 to 60000 commands.

[1] ML Repository: http://archive.ics.uci.edu/ml/datasets/UNIX+User+Data

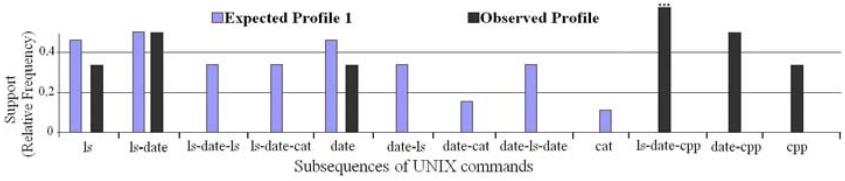

Fig. 3. Observed and Expected Comparison Example

– **Set of 50 UNIX Users:** Data[2] used in the masquerade-detection studies done by Schonlau et al. [9]. In Schonlau research, commands from other users are interspersed as masqueraders data. In our research, the 50 users data are used without these commands interspersed. Each user file contains 15000 commands.

In both cases, the data is drawn from *tcsh* history files and pre-processed to remove filenames, user name, directory structures, etc. Command names, flags, and shell meta characters have been preserved. However, this analysis is only based on two fields: *Command name* and *User Identification*. Thus, a user is identified by a set of commands concatenated by date order; for example the first 10 commands of the *User1* in the *50 Users set* are: *cpp, sh, xrdb, cpp, sh, xrdb, mkpts, env, csh, csh.*

4.1 Experimental Design

In order to measure the performance of the proposed classifier using the above data, the well-established technique cross-validation is used. For this research, **10-fold cross-validation** is used: We remove a 10% of the commands from the initial data of each user and the corresponding *distributions* are calculated (*Training Distributions*). Then, the portion of data originally taken out of each user data is analyzed and its corresponding distribution is created (*Test Distribution*). Using the proposed statistical method, these distributions are compared and the user is classified. As 10-fold cross validation is used, this process is repeated 10 times per user.

The number of UNIX commands analyzed per user is very relevant for the classification result. Therefore, we have performed several experiments with different number of UNIX commands (50, 100, 500, 1000 and 5000) per user. These commands are selected from the last commands typed by a user. Also, in the phase of behavior model creation, the length of the subsequences in which the original sequence is segmented (used for creating the *trie*) is a relevant parameter: Using a longer length, the time consumed for creating the *trie* and the number of relevant subsequences in the corresponding distribution increase drastically. In the experiments presented in this paper, 3 different segmentation values for the sequence (subsequence lengths) are evaluated: 3, 5 and 10.

[2] Schonlau web page: http://www.schonlau.net/intrusion.html

4.2 Results

In this research, a UNIX command sequence (*Test Distribution*) is classified into the user behavior (*Training Distribution*) with the smallest deviation. Also, the classification process generates a ranked list with the most likely users at the top. Although there are users whose behavior is quite similar, in the proposed experiments, the classification is correct only if the user who typed the sequence of commands to classify holds the first position of the ranking list.

The results are listed in Table 1. The classification rate is the ratio of the number of correct classifications made and the standard deviation measures the dispersion of the classification results according to the obtained ranking list.

Table 1. Classification Results using ABCD. 9 and 50 Users.

		ABCD Classifier Results			
		Set of 9 UNIX Users		**Set of 50 UNIX Users**	
Number of commands	Subseq Length	Classification rate %	Standard Deviation	Classification rate %	Standard Deviation
50	3	80,00	1,40	48,20	8,99
	5	78,89	1,34	48,80	7,73
100	3	80,00	0,96	53,40	8,42
	5	76,67	1,08	51,40	9,81
	10	78,89	0,83	54,80	6,99
500	3	90,00	1,08	64,00	9,16
	5	91,11	1,27	64,20	10,17
	10	86,67	1,49	63,80	12,48
1000	3	87,78	1,53	72,00	10,14
	5	87,78	1,30	71,20	10,49
	10	81,11	1,84	69,00	11,69
5000	3	85,56	1,23	75,80	12,05
	5	87,78	1,30	76,60	12,26
	10	84,40	1,54	75,00	12,64

We can see from the *Set of 9 Users* results (Table 1) that even with 50 commands (45 per training and 5 per testing), the classification rate is very high (around 80%). The results obtained with different subsequence lengths for creating the *trie* (3, 5 and 10) show that the higher classification rates are not obtained using a higher length. The higher classification rate is usually obtained using subsequences of length 5; this number determines the number of commands considered as dependent for a UNIX user.

According to the *Set of 50 Users* results (Table 1), the classification rate is smaller because of the high number of users to classify. In this case, this rate increases considerably with increasing the number of commands for training and testing. Using 5000 commands (4500 for training and 500 for testing), the classification rate is higher than 75% and if we can get more than 900 commands for training, the classification rate is higher than 70%.

5 *ABCD* vs. HMMs

Recent researches have demonstrated the effectiveness of *Hidden Markov Models* (HMMs) for information extraction and they are very used in speech recognition. However, HMMs can efficiently deal with time-sequential data and can provide time-scale invariability as well as learning capability for recognition. Therefore, HMMs are also used in the environment we propose in this research. HMMs have been used for recognizing automated robot behaviors [20] and recently, for behavior understanding from video streams [21]. In addition, Lane [22] demonstrates the use of HMMs for user profiling in the domain of anomaly detection using a data set very similar to the set used in our research. An improved HMM-based method for this purpose is proposed in [23].

For this reason, to evaluate the results shown in the previous section, we compare them with a **classifier based on HMMs**. A HMM is a finite set of states, each of which is associated with a probability distribution [24]. Transitions among the states are governed by a set of probabilities called transition probabilities. In a particular state an observation can be generated, according to the associated probability distribution (it is only the observation, not the state visible to an external observer).

To define a HMM completely, the following elements are needed: 1) Number of observation symbols in the alphabet, M. 2) Number of states of the model, K. 3) A state transition probabilities matrix, A. 4) A probability distribution in each of the states, B. 5) The initial state distribution, Π.

In order to classify the behavior of UNIX users using a HMM-based method, **a HMM is created for each user** as follows: The number of observation symbols (M) is the number of different commands typed by the user. The number of states of the model (K) is an open question in the use of HMMs for modeling but its choice is important because it affects the potential descriptiveness of the HMM. In our research, according to the study done in [22] and in order to compare the *ABCD* and HMM results; the number of states of a HMM corresponds with the *subsequence length* used in *ABCD* for creating the *trie*.

The toolkit Umdhmm [25] was used to create each HMM (UNIX user behavior model) from the corresponding training data files. After creating the HMMs, the ***Forward Algorithm*** is used to calculate the probability of an observed UNIX user sequence (*Test HMM*) given a user model (*Training HMM*). The sequence of commands is classified into the HMM with the highest likelihood.

Table 2 shows the results using a classifier based on HMMs and using the same data than in the previous experiments (Section 4). These results show that with a low number of commands for training, a classifier based on HMMs gets a low classification rate. Thus, using HMMs we need a high number of commands to get similar results to the obtained using *ABCD*. However, creating the user models with more than 5000 commands, the classification rate is usually *a bit* better using HMMs. Even so, the difference in the classification rate between *ABCD* and HMMs in the *Set of 50 users* is very significant. It is remarkable the high classification rate obtained by *ABCD* using a low number of commands (for training and classifying). For areas such as computer intrusion detection,

Table 2. Classification Results using HMMs. 9 and 50 Users.

		HMMs Classifier Results			
		Set of 9 UNIX Users		Set of 50 UNIX Users	
Number of commands	Subseq Length	Classification rate %	Standard Deviation	Classification rate %	Standard Deviation
50	3	52,22	2,23	30,40	14,08
	5	54,44	2,06	32,40	14,72
	10	54,44	2,08	34,80	15,02
100	3	64,44	1,49	39,40	8,72
	5	61,11	1,53	40,00	8,58
	10	62,22	1,60	40,40	8,94
500	3	63,33	1,22	42,20	6,19
	5	68,89	1,30	48,20	6,03
	10	66,67	1,26	51,20	5,86
1000	3	63,33	1,20	46,20	4,69
	5	68,89	1,32	49,20	4,55
	10	66,67	1,09	53,20	4,47
5000	3	80,00	1,05	54,20	3,89
	5	82,22	0,90	58,20	3,53
	10	88,89	0,97	62,20	3,45

this aspect is really important because the detection can be done when the user only has typed a few commands and the set of users is small.

6 *Future Work: ABCD Adaptative*

A widely acknowledged challenge in the *ABCD* is how to accurately profile a user while his/her behavior changes constantly. Thus, a user profile should be frequently revised to keep it up to date. To solve this problem, we propose a technique used by Angelov and Zhou. [26] for analyzing the quality of the rule base in an on-line fuzzy system. This technique uses the **moment when the information is obtained**.

Applying this technique in *ABCD*, the subsequences typed by a user are indexed with a number that indicates the *moment* they were read. This value can be considered as an integer from 1 (the first subsequence read) to the number of subsequences read. Using this value, the *Age* of a subsequence can be calculated. This *Age* value indicates how old a subsequence stored in a user profile is. The formula for calculating this value is shown in Equation 1.a.

$$\textbf{a.}\ \ Age_S(t) = t - \frac{\sum_{i=1}^{N_s(t)} I_s(i)}{N_s(t)}\ ; \qquad \textbf{b.}\ \ \overline{Age(t)} = \frac{1}{R}\sum_{j=1}^{R} Age_i(t) \tag{1}$$

where t is the current time instant; s represents a certain subsequence; $Age_s(t)$ denotes the *Age* of the subsequence s in the moment t; $N_s(t)$ is the number

of times the subsequence s was read until the moment t and $I_s(i)$ denotes the *moment* of the subsequence s when it was read for i^{th} time.

Using this value, the distribution of subsequences that represents a user profile can be **updated on-line**. Thus, the *Age* of each subsequence can be calculated and compared with the *mean Age* that is determined in Equation 1.b.

These values can be used for removing older subsequences that were used by a user but during a long period of time they have been omitted. Also, major shifts in the user behavior can be detected using the *Age* value.

7 Conclusions

This paper presents an approach (*ABCD*) for profiling and classifying computer users from a command-line interface. The sequence of commands typed by user is segmented and stored in a *trie* data structure, and the relevant subsequences are evaluated by using a frequency-based method. Then, a user profile is represented by a distribution of relevant subsequences and a modification of the *Chi-square Test* for two samples is proposed for recognition of users. In addition, as the behavior of a user can change constantly, we also propose a technique to updated these profiles by calculating the *Age* of each subsequence and removing the no relevant ones.

ABCD has been evaluated with real-data analyzing two different data sources which have different numbers of users: 9 and 50 UNIX users. A large set of experiments were conducted and the obtained results by using the *ABCD* are very satisfactory. The comparison of *ABCD* with a classifier based on HMMs shows that the proposed technique is more suitable in the environment evaluated, mainly when the training data is small. These results are very encouraging because analyzing few commands, a user (and his/her behavior) can be recognized and then, different actions in the computer system, (such as to monitor, analyze and detect abnormalities, assist the user, predict his/her future actions or detect masqueraders) can be executed. However, *ABCD* is generalizable and it could be evaluated in many other different domains (the only constraint is that the behavior can be represented as a sequence of commands or events).

Finally, if we want to analyze hundreds (or thousands) of users, *ABCD* can be easily modified for clustering users with similar profiles. This aspect could be implemented using *Evolving Systems* [27] and it is proposed for future work.

References

1. Mulcahy, N.J., Call, J.: Apes save tools for future use. Science 312(5776), 1038–1040 (2006)
2. Hackos, J.T., Redish, J.C.: User and Task Analysis for Interface Design. Wiley, Chichester (1998)
3. Spiliopoulou, M., Faulstich, L.C.: Wum: A web utilization miner. In: Proceedings of EDBT Workshop WebDB 1998, pp. 109–115. Springer, Heidelberg (1998)
4. Wexelblat, A.: An environment for aiding information-browsing tasks. In: Proc. of AAAI Spring Symposium on Acquisition, Learning and Demonstration: Automating Tasks for Users. AAAI Press, Menlo Park (1996)

5. Macedo, A.A., Truong, K.N., Camacho-Guerrero, J.A., da GraÇa Pimentel, M.: Automatically sharing web experiences through a hyperdocument recommender system. In: HYPERTEXT 2003, pp. 48–56. ACM, New York (2003)
6. Godoy, D., Amandi, A.: User profiling for web page filtering. IEEE Internet Computing 9(4), 56–64 (2005)
7. Pepyne, D.L., Hu, J., Gong, W.: User profiling for computer security. In: Proceedings of the American Control Conference, pp. 982–987 (2004)
8. Coull, S.E., Branch, J.W., Szymanski, B.K., Breimer, E.: Intrusion detection: A bioinformatics approach. In: Omondi, A.R., Sedukhin, S.G. (eds.) ACSAC 2003. LNCS, vol. 2823, pp. 24–33. Springer, Heidelberg (2003)
9. Schonlau, M., Dumouchel, W., Ju, W.H., Karr, A.F.: Theus, Computer Intrusion: Detecting Masquerades. Statistical Science 16, 58–74 (2001)
10. Anderson, J.: Learning and Memory: An Integrated Approach. John Wiley and Sons, New York (1995)
11. Horman, Y., Kaminka, G.A.: Removing biases in unsupervised learning of sequential patterns. Intelligent Data Analysis 11(5), 457–480 (2007)
12. Lane, T., Brodley, C.E.: Temporal sequence learning and data reduction for anomaly detection. In: CCS 1998, pp. 150–158. ACM, New York (1998)
13. Riley, P., Veloso, M.M.: On behavior classification in adversarial environments. In: DARS, pp. 371–380
14. Iglesias, J.A., Ledezma, A., Sanchis, A.: Sequence classification using statistical pattern recognition. In: Berthold, M.R., Shawe-Taylor, J., Lavrač, N. (eds.) IDA 2007. LNCS, vol. 4723, pp. 207–218. Springer, Heidelberg (2007)
15. Fredkin, E.: Trie memory. Comm. ACM 3(9), 490–499 (1960)
16. Kaminka, G.A., Fidanboylu, M., Chang, A., Veloso, M.M.: Learning the sequential coordinated behavior of teams from observations. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS, vol. 2752, pp. 111–125. Springer, Heidelberg (2003)
17. Iglesias, J.A., Ledezma, A., Sanchis, A., Kaminka, G.A.: Classifying efficiently the behavior of a soccer team. In: Burgard, W., et al. (eds.) Intelligent Autonomous Systems 10. IAS-10, pp. 316–323 (2008)
18. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Eleventh International Conference on Data Engineering, Taipei, Taiwan, pp. 3–14 (1995)
19. Blake, C., Newman, D.J., Hettich, S., Merz, C.: UCI repository of machine learning databases (1998)
20. Han, K., Veloso, M.: Automated robot behavior recognition applied to robotic soccer. In: IJCAI 1999 Workshop on Team Behaviors and Plan Recognition (1999)
21. Chung, P.-C., Liu, C.-D.: A daily behavior enabled hidden markov model for human behavior understanding. Pattern Recognition 41(5), 1572–1580 (2008)
22. Lane, T.: Hidden Markov Models for Human-computer interface modeling. In: Proceedings of IJCAI 1999 Workshop on Learning About Users, pp. 35–44 (1999)
23. Lane, T., Brodley, C.E.: An empirical study of two approaches to sequence learning for anomaly detection. Machine Learning 51(1), 73–107 (2003)
24. Bengio, Y.: Markovian models for sequential data. Neural Computing Surveys 2, 129–162 (1999)
25. Kanungo, T.: Umdhmm: A hidden markov model toolkit. In: Extended Finite State Models of Language. Cambridge Univ. Press, Cambridge (1999)
26. Angelov, P., Zhou, X.: Evolving fuzzy-rule-based classifiers from data streams. IEEE Transactions on Fuzzy Systems 16(6), 1462–1475 (2008)
27. Angelov, P.: Rule-based Models: A Tool for Design of Flexible Adaptive Systems. Springer, Heidelberg (2002)