An Adaptive User Interface Based on Personalized Learning

Jiming Liu, Chi Kuen Wong, and Ka Keung Hui, *Hong Kong Baptist University*

This adaptive user interface provides individualized, just-in-time assistance to users by recording user interface events and frequencies, organizing them into episodes, and automatically deriving patterns. It also builds, maintains, and makes suggestions based on user profiles.

The design and complexity of a software system's user interface largely determines the ease with which users can effectively operate that system. We have designed a dynamic, seamlessly personalized adaptive user interface: it reacts to different situations and requirements, and it learns individual users' behavior patterns as well as styles. Once the AUI records the events of human-computer interaction and discovers patterns of user behavior, it can provide just-in-time assistance by predicting a user's most likely plan and then performing part of the plan on the user's behalf. It also manipulates the software system semiautonomously, thus reducing the intervention required.

The adaptive user interface

Our approach is based on *episodes identification and association*. (See the "Related Work" sidebar for information on others' research.) EIA enables the interface to recognize user action plans by tracing and analyzing a user's action sequences. Implementing this approach involves five important issues:

1. *Observing the interaction between a user and a software application*. We might lose data because of limitations in the types of events we can perceive. So, to extract as much useful information about a user's intentions and needs as possible, we must identify various low-level interface events from available data.
2. *Identifying different episodes from the actions we observe in user–computer interaction*. We must formulate several important classes from user interface data—for example, keyboard typing, menu selection, and button clicking. We consider each action a *basic episode*. These observable clues will help us recognize a user's intention.
3. *Recognizing user behavior patterns*. To reveal hidden patterns in the streams of events we retrieve, we need a language or algorithms for representing and computing the events into associated patterns. Moreover, the AUI should be able to compose new events from previously modeled events and build or modify the transformation functions for the modeled events.
4. *Adaptively helping users according to recognized user plans*. When the interface recognizes that the user is about to execute a certain plan, it should offer assistance. The user can configure the interface with his or her preferences on where and how to display the proposed help.
5. *Building user profiles that will enable personalized interactions*. Profiles store both user-defined preference information and system-detected user behavior patterns. The system updates a profile whenever it detects a change in user behavior patterns.

We focus here on the formulations and algorithms for identifying episodes and recognizing patterns (that is, the second and third issues).

Our approach discovers the rules that can best describe and predict user behavior by finding frequently occurring episodes in user action sequences. This approach is partly inspired by earlier work on methods for discovering frequent episodes, called Window Episode and Minimal Occurrence Episode.[1] To reliably detect patterns from a small number of observed sequence data samples, we induce implication relations—that is, we construct dependence

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

Related Work

Earlier adaptive user interfaces, including Letizia[1] and Let's Browse,[2] let users browse the World Wide Web by tracking user behavior and anticipating items of interest. They analyze user behavior by matching the keywords in Web documents.

To date, AUIs primarily use the following computational approaches: Bayesian network, mixed-initiative, model-based, and programming by example.

For instance, the Lumiere project uses Bayesian network and influence diagrams to infer users' goals by considering their background, actions, and queries.[3] Microsoft Office Assistant is the result of porting Lumiere components to Microsoft Office; it employs Bayesian user models to infer user goals. DeepListener is a spoken command and control development environment that also employs Bayesian networks to model the uncertainty in a user's goals and utterances over time.[4] The LookOut system enhances human-computer interaction when the user schedules meetings through automated services with direct access to Microsoft Outlook.[5] It automatically identifies messages and tries to help users review their calendars and arrange appointments. Its mixed-initiative approach is designed to solve a user's scheduling problems—where it is assumed that intelligent services and a user can collaborate efficiently to achieve the user's goals.

The model-based approach and programming-by-example are popular approaches for designing AUIs. For instance, developers have used the model-based approach to design user interfaces for electronic patient records[6] and automatic help generation.[7] Generally speaking, the model-based approach does not involve a learning component. It predicts a user's plan largely on the basis of an accurate user model.[8] So, selecting the right set of observable properties in the user's world becomes crucial. In practice, this task can be difficult and costly.

Dynamic Macro and SMARTedit[9] are two systems that automate repetitive text-editing tasks using programming-by-example. Although using macro recorders in a programming-by-example system enables automatic code generation without too much user intervention, it still requires substantial user effort to demonstrate to the system the programming task to automate. Also, conventional macros can only replay the exact sequence of actions recorded. In many cases, they are too specific to be reused.

References

1. H. Lieberman, "Letizia: An Agent That Assists Web Browsing," MIT Media Lab, 1995, http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia-AAAI/Letizia.html.
2. A. Vivacqua, H. Lieberman, and N.V. Dyke, "Let's Browse: A Collaborative Web Browsing Agent," MIT Media Lab, 1997, http://lieber.www.media.mit.edu/people/lieber/Lieberary/Lets-Browse/Lets-Browse.html.
3. E. Horvitz et al., "The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users," *Proc. 14th Ann. Conf. Uncertainty in Artificial Intelligence* (UAI 98), Morgan Kaufmann, 1998, pp. 256–265; ftp://ftp.research.microsoft.com/pub/ejh/lum.pdf.
4. E. Horvitz and T. Paek, "Harnessing Models of Users' Goals to Mediate Clarification Dialog in Spoken Language Systems," *Proc. 8th Int'l Conf. User Modelling*, 2001; www.research.microsoft.com/research/dtg/horvitz/umdl.htm.
5. E. Horvitz, "Uncertainty, Action, and Interaction: In Pursuit of Mixed-Initiative Computing," *IEEE Intelligent Systems*, vol. 14, no. 5, Sept./Oct. 1999, pp. 17–20.
6. A.R. Puerta, "Design of Adaptive User Interfaces for Electronic Patient Records," *Proc. CHI 98 Workshop User Interfaces for Computer-Based Patient Records*, 1998; www.diamondbullet.com/cpr/paper-puerta.html.
7. R. Moriyon, P. Szekely, and R. Neches, "Automatic Generation of Help from Interface Design Models," *Proc. SIGCHI Conf. Human Factors in Computing Systems: Celebrating Human Performance* (CHI 94), ACM Press, 1994, pp. 235–241.
8. R. Opperman, "Adaptively Supported Adaptivity," *Int'l J. Human-Computer Studies*, vol. 40, no. 3, Mar. 1994, pp. 455–472.
9. H. Lieberman, ed., *Your Wish Is My Command: Programming by Example*, Morgan Kaufmann, 2001.

relationships between ordered events based on statistical testing.[2,3]

Figure 1 shows an AUI schematic. The interface tracks interaction between a user and an application software system and tries to identify consistent user behavior patterns. On the basis of these patterns, it can interact with the software on the user's behalf whenever the user authorizes the interface to do so. The interface might take a while to detect user patterns if it starts from scratch every time the user uses the software. Alternatively, the AUI can build a user profile that includes all the patterns recognized in past interaction sessions. As it detects new patterns, it updates the profile.

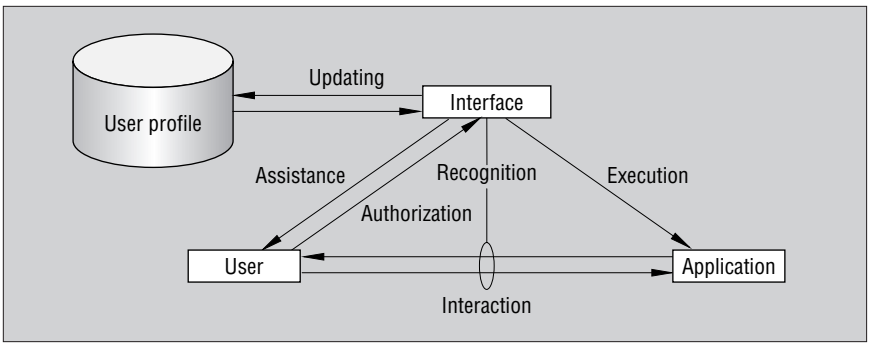


Figure 1. A schematic of an adaptive user interface.

Figure 2 presents a basic AUI architecture comprising four key components: *event capture*, *event identification*, *user pattern recognition*, and *user intention prediction*. Event capture tracks the interaction between a user and an application. Each action is represented with an identification number

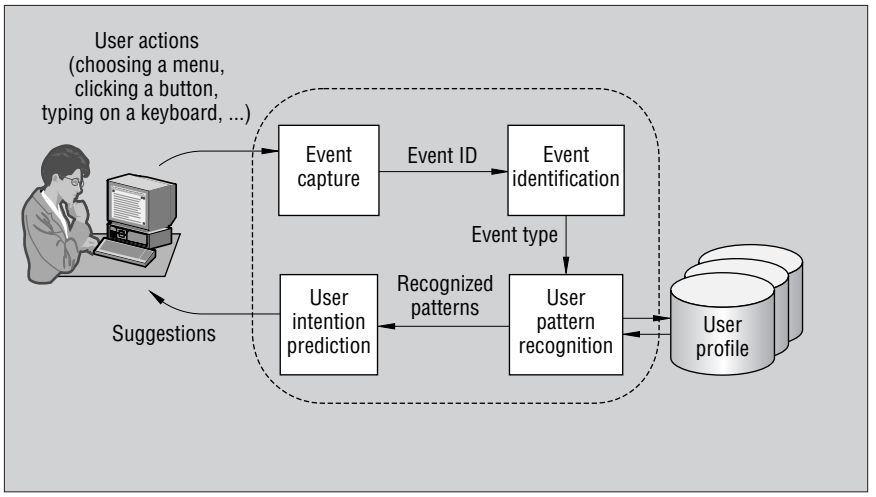


Figure 2. An adaptive user interface architecture.

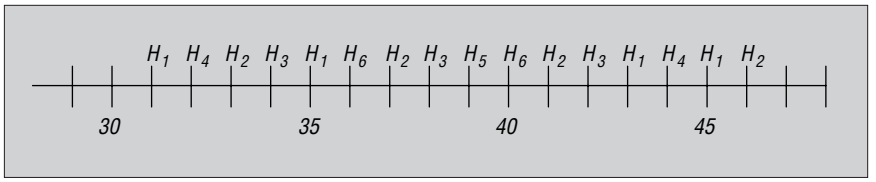


Figure 3. An example episode sequence.

mapped onto a table to identify the action type. User pattern recognition predicts a user's intention by matching action sequences with the patterns stored in his or her profile. It also tries to detect new user patterns and adds them to the profile. If this component finds a pattern match, user intention prediction instantiates the pattern with the current parameters and displays suggestions to the user.

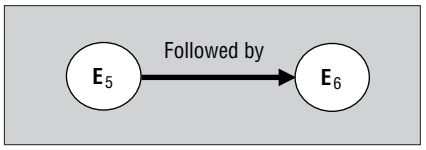


Figure 4. An example serial episode E_s.

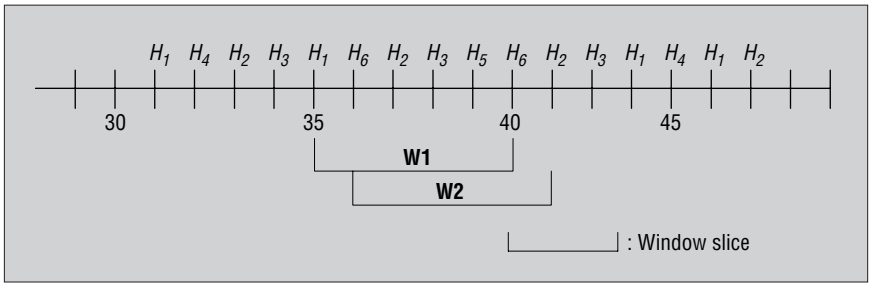


Figure 5. Two partially overlapping windows of width 6.

Implementation techniques

We present five computational techniques for implementing EIA.

Episodes discovery

To detect user behavior patterns, the interface must be able to track all basic interface events. Here we distinguish two types of interface events: text input events, where each word input is a basic episode, and mouse click events, identified by mapping an event ID to a corresponding button through a mapping table. Some events will need additional processes to retrieve their attribute's value. For example, the Font Size event's attribute is a numeric value.

Specifically, the interface captures and represents user–application interactions as a sequence (**S**) of episodes within a time interval. The technique expresses each episode **E** as a pair (**H**, *t*), where **H** is an interface event type and *t* is the time of the episode's occurrence. Some interface event types can contain a vector to store their associated attributes.

We define **S** as (**s**, T_s, T_e), where **s** is an episode sequence and the sequence's starting time $T_s <$ its ending time T_e. That is,

$$\mathbf{s} = \{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_n\} = \{(\mathbf{H}_1, t_1), (\mathbf{H}_2, t_2), \ldots, (\mathbf{H}_n, t_n)\}$$

where $T_s \le t_i < T_e$, $t_i < t_{i+1}$, and $\forall i = 1, \ldots, n$.

Figure 3 shows an example of an episode sequence $\mathbf{S} = (\mathbf{s}, 31, 46)$, where $\mathbf{s} = \{(\mathbf{H}_1, 31), (\mathbf{H}_4, 32), (\mathbf{H}_2, 33), \ldots, (\mathbf{H}_2, 46)\}$.

Figure 4 shows an example of a serial episode $\mathbf{E}_s$ in which the *post-episode* $\mathbf{E}_6$ follows the *pre-episode* $\mathbf{E}_5$.

The Window Episode method

To discover frequent episodes within an episode sequence, we define a time window W as a slice of an episode sequence; it divides a long sequence into a number of shorter sequences. Window slices can partially overlap on an episode sequence, as Figure 5 shows.

Specifically, we define **W** as (**w**, t_s, t_e), where **w** is an episode sequence, t_s is the sequence's starting time, t_e is its ending time, and $T_s \le t_s < t_e = T_e$. In other words,

$$\mathbf{w} = \{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_n\} = \{(\mathbf{H}_1, t_1), (\mathbf{H}_2, t_2), \ldots, (\mathbf{H}_n, t_n)\}$$

where $t_s \le t_i < t_e$, $t_i < t_{i+1}$, and $\forall i = 1, \ldots, n$.

Moreover, we call the time difference $t_e - t_s + 1$ the width of a window **W** and write it as *width*(**w**). The width must be at least 2 units and no more than the length of an episode sequence. After a long sequence is sliced, many windows, defined as **W**(**s**, *width*(**w**)), will have the same width.

Figure 5 illustrates the window concept. The window **W1** is described as **W1** = (**w**, 35, 40), where $\mathbf{w} = \{(\mathbf{H}_1, 35), (\mathbf{H}_6, 36), (\mathbf{H}_2, 37), (\mathbf{H}_3, 38), (\mathbf{H}_5, 39), (\mathbf{H}_6, 40)\}$, *width*(**w**) = 6, and **W**(**s**, *width*(**w**)) is a set of windows (that is, {**W1**, **W2**, …}) of the same width.

Frequency

An episode's *frequency* is the number of windows in which the episode occurs. That means, given an episode sequence **s** and a

window width limited by *win*, the frequency of an episode **E** in **s** is

$$fr(\mathbf{E}, \mathbf{s}, win) = |[\boldsymbol{w} \in \mathbf{W}(\boldsymbol{s}, win) \mid \mathbf{E} \text{ occurs in } \mathbf{w}]|.$$

To be a frequent episode, it must pass two tests. The first is the frequency threshold (min_*fr*) test

$$\frac{fr(\mathbf{E}_a \Rightarrow \mathbf{E}_b, \mathbf{s}, win)}{fr(\mathbf{E}_a, \mathbf{s}, win)} \geq \min\_fr,$$

where $(\mathbf{E}_a \Rightarrow \mathbf{E}_b)$ is a *composite* episode consisting of two sequential episodes, $\mathbf{E}_a$ and $\mathbf{E}_b$. The second is the confidence threshold (min_*conf*) test

$$\frac{fr(\mathbf{E}_a, \mathbf{s}, win)}{|\mathbf{W}(\mathbf{s}, win)|} \geq \min\_conf.$$

Recursive composition of frequent episodes

These formulations and tests enable the interface to easily find many frequent episodes, but they would not be particularly useful if the episodes it found dealt only with pairwise individual interface events. What is most desirable is that the interface discover longer, more complex frequent episodes. For instance, one or two episodes might imply another episode to follow, two might imply another two, and so on. To achieve this, we recursively apply the Window Episode method by combining newly modeled episodes with previously found episodes (see Figure 6). After we get a frequent composite episode $\mathbf{E}_a \Rightarrow \mathbf{E}_b$ (that is, episode $\mathbf{E}_a$ is followed by episode $\mathbf{E}_b$), it will act as a pre-episode and imply others.

Implication relations

Sometimes, the interface must discover frequent episodes with a limited number of sequence samples. We use an implication induction algorithm to handle this situation.[2,3] Specifically, for each sequence relation $\mathbf{E}_a \Rightarrow \mathbf{E}_b$, this algorithm computes the lower bound of a $(1 - \alpha_c)$ confidence interval around a measured conditional probability $\mathbf{p}_{\min\_fr}$, where α_c is the maximum probability of having a random error (also called *alpha error*).

Suppose that N_{error} number of observations have violated sequence relation $\mathbf{E}_a \Rightarrow \mathbf{E}_b$. So, on the basis of the binomial distribution $Bin(N_{\text{error}}, p_{\min\_fr})$, the algorithm tests whether the probability of errors is less than an acceptable threshold:

$$p(x \leq N_{\text{error}}) < \alpha_c,$$

where α_c is the alpha error of this conditional probability test.

The Personalized Word Assistant

To demonstrate the feasibility of applying EIA to personalize interface assistance, even under an existing application's interface constraints, we developed a personalized AUI to augment Microsoft Word's originally hard-coded interface functionality. We call this new interface the Personalized Word Assistant.

Although MS Word provides phrase completion and simple format automation, these features' capabilities and their implementation are different from those in our EIA method. EIA is based on frequent episodes rather than histories or lookup tables. Episodes, basic or composite, must pass a frequency threshold test and a confidence threshold test before we can use them to predict a user's intention. Instead of building a personalized user profile before user interaction starts, the EIA-based AUI can gradually learn the useful frequent episodes on the fly.

Personalized Word Assistant can provide two levels of assistance. At the word level, it makes phrase associations, treating one or more associated words as an episode, and a sequence of unassociated words as an event sequence. With a *bag-of-word* text document representation (a vector of word pairs and their occurrence frequency), PWA readily finds out the word (episode) frequency counts. It discovers associations among words by using the EIA algorithms described earlier.

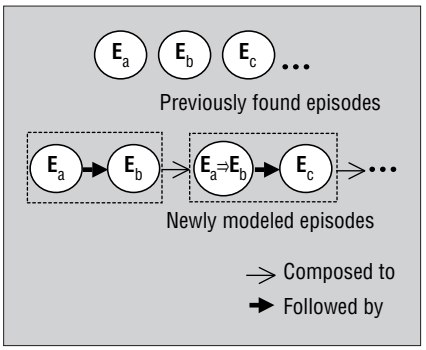


Figure 6. Recursive composition of frequent episodes.

The example in Figure 7 illustrates phrase association. Each word acts as an episode. If an episode contains one word (for example, "computer"), we call it a 1-gram. If it contains two words (for example, "computer" and "science"), we call it a 2-gram. The *n*-gram representation can consider up to five words in an episode. PWA will help a user by suggesting the next word or phrase on the basis of what the user has typed.

In paragraph-level assistance, PWA concentrates on automating formatting and finds consistent formats in paragraphs. If a user changes some paragraph formatting and then performs the same operations elsewhere, PWA detects a format change pattern and discovers a frequent episode, represented as $\mathbf{F}_1 \Rightarrow \mathbf{F}_2$. For instance, changing from format $\mathbf{F}_1$ to format $\mathbf{F}_2$ requires five operations (see Figure 8). Considering formatting operations as episodes in EIA, "font" ⇒ "bold" ⇒ "italic" ⇒ "font size" ⇒ "alignment," the

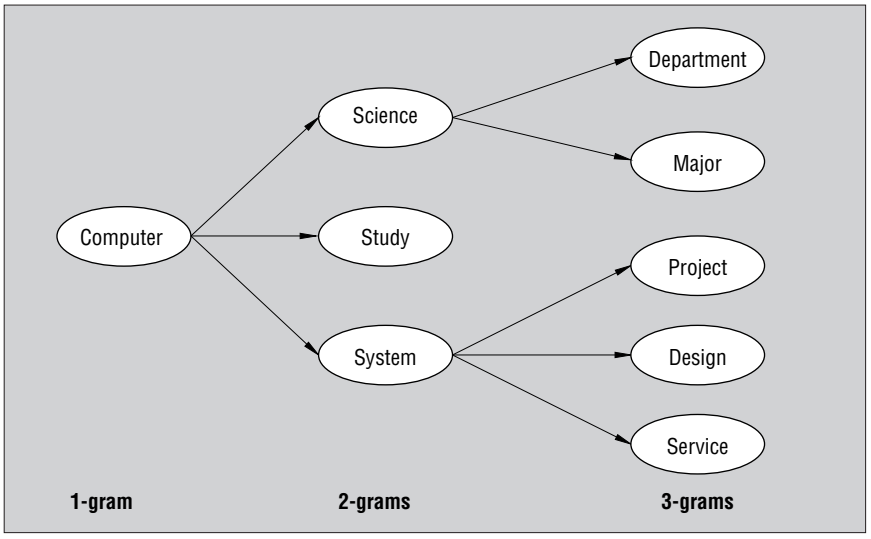


Figure 7. A phrase association example.

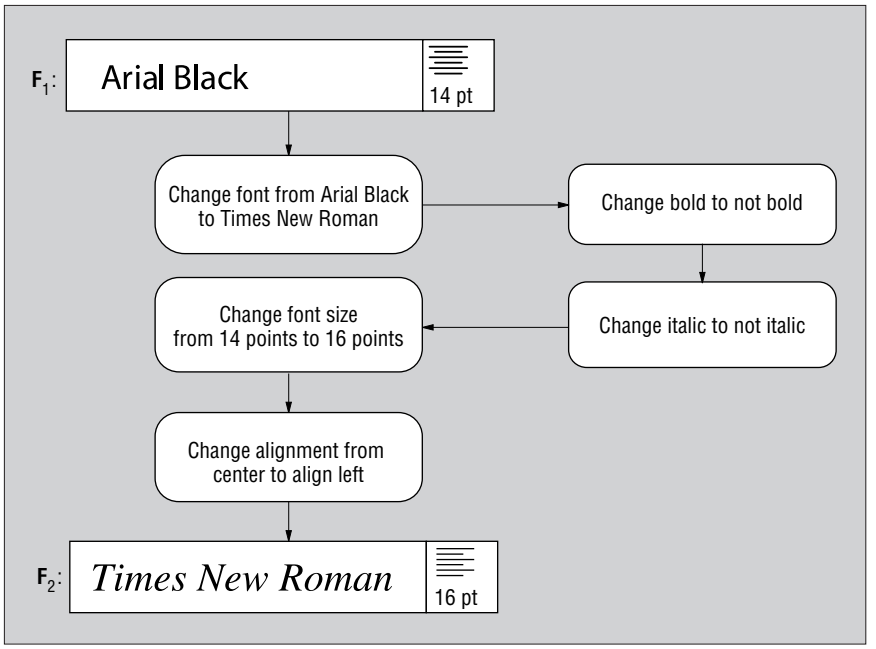


Figure 8. A paragraph formatting example.

interface will be able to find an association between $\mathbf{F}_1$ and $\mathbf{F}_2$. With those detected frequent episodes, the user can instruct the interface to automatically carry out the corresponding consistent format changes in other paragraphs.

PWA's windows and options are easy to use without training. Through dialog boxes, the user can select preferences for threshold values and the minimum and maximum number of words in a phrase for association. The user can also enable and disable the interface at any time during an interaction.

Evaluation

To examine EIA's effectiveness and characteristics, we conducted two experiments.

Usability

Our first study examined the usability of PWA's automation capability. The seven subjects were nontechnical staff and postgraduate students from the Department of Computer Science. We asked them to determine the types of documents they would write and to provide approximately 12 to 15 similar old documents for PWA to build its personalized user profiles. We set the frequency threshold (min_*fr*) and confidence threshold (min_*conf*) for phrase association at 0.25 and 0.001, respectively, and at 0.2 and 0.01 for format automation. They then typed related documents with PWA's help. During their typing, we recorded the number of PWA's suggestions and the number of suggestions the users accepted. We found that the acceptance rates for phrase association and format automation were 75 percent and 86 percent (the system made an average of 175 and 83 suggestions per user, respectively).

At the end of the experiment, we administered a questionnaire to each user to collect feedback about the quality of PWA suggestions and the user interface. The users rated the quality measures on a seven-point range, from 1 for very bad to 7 for very good. The phrase association rating was 5.78, and the format automation rating was 5.69. Their ratings on the AUI were 5.71 and 6.11, respectively. You might argue that, for most short phrases, removing your hands from the keyboard to click on a phrase completion takes more effort than simply typing the phrase. The validity of this argument depends on the user's typing skill: A slow typist might prefer clicking a button rather than typing a few words. For skilled typists, we can enhance our user interface so that they can select a suggested phrase with one keystroke (for example, by hitting one function key).

Productivity

The next study evaluated whether using PWA would increase users' productivity in typing documents, measured in terms of the time and operation steps required. We randomly divided 14 third-year computer majors into two groups. One group typed documents with PWA; the other served as a control group. PWA automatically recorded typing times and the number of operation steps. The frequency and confidence threshold settings were the same as those in the usability experiment. We found that the experimental group's productivity was much higher than the control group's. On average, the experimental group completed the task in 34 percent less time and with 58 percent fewer steps.

Episode-based learning is simpler than traditional approaches to user adaptation. Our method supports automated learning and personalized adaptation. However, it only records formulations sequentially without concurrency.

Our two pilot studies showed that users readily accept PWA's suggestions and type more productively using it. These initial results of empirical validation are encouraging and show that our approach is theoretically sound and practically feasible. However, a definite conclusion based on a large sample of subjects will be a topic for further research.

Selecting input characters with a stylus in handheld devices is slow and difficult. Our EIA-based approach to building AUIs can help automate repetitive typing and thus reduce the amount of time and number of operations needed for composing text. We can also apply the method to other domains, such as spreadsheet and computer-aided design applications, to learn a user's habits and anticipate repetitive tasks for possible automation.

Some issues deserve further investigation. First, for the present methods and prototypes, all threshold values and the length of episode sequences are fixed for all users. The interface might perform more effectively if these values were assigned, or dynamically adjusted, according to a user's behavior patterns. So, it would be desirable to build in (model- or profile-based) self-adapting mechanisms for fine-tuning learning parameters and for filtering and constructing variable-length episode sequences.

Second, our system currently stores all detected user patterns in user profiles. However, some user interests or behavior patterns might be temporary, changing over time. Therefore, a self-adapting mechanism, such as a forgetting policy based on the frequency of user rejection of system suggestions, would be desirable for balancing short-term and long-term learning.

Finally, we have validated our approach with a specific application. We plan to extend and implement this approach in an application-independent context—for example, at the window manager level. Then we could easily configure such a generic tool to work with a variety of software applications. ■

References

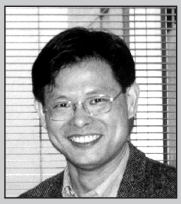
1. H. Mannila and H. Toivonen, "Discovering Generalized Episodes Using Minimal Occurrences," *Proc. 2nd Int'l Conf. Knowledge Discovery and Data Mining* (KDD 96), AAAI Press, 1996, pp. 189–194.

2. J. Liu and M.C. Desmarais, "A Method of Learning Implication Networks from Empirical Data: Algorithm and Monte-Carlo Simulation-Based Validation," *IEEE Trans. Knowledge and Data Eng.*, vol. 9, no. 6, Nov./Dec. 1997, pp. 990–1004.

3. M.C. Desmarais, D.A. Maluf, and J. Liu, "User-Expertise Modeling with Empirically Derived Probabilistic Implication Networks," *Int'l J. User Modeling and User-Adapted Interaction*, vol. 5, nos. 3–4, 1996, pp. 283–315.

For more information on this or any other computing topic, visit our Digital Library at http://computer.org/publications/dlib.

The Authors



Jiming Liu is the head of the Computer Science Department at Hong Kong Baptist University and the editor in chief of *Web Intelligence and Agent Systems: An International Journal* and *Annual Review of Intelligent Informatics*. His books include *Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation*; *Multi-Agent Robotic Systems*; and *Web Intelligence*. He received his PhD in electrical engineering from McGill University. He is a senior member of the IEEE and a member of the ACM and AAAI. Contact him at Hong Kong Baptist Univ., Computer Science Dept., 224 Waterloo Rd., Kowloon Tong, Hong Kong; jiming@comp.hkbu.edu.hk; http://robotics.comp.hkbu.edu.hk/~jiming.

Chi Kuen Wong is an assistant professor of computer science at Hong Kong Baptist University. His research interests include intelligent interactive learning environments, Web-based assessments, and human-computer interfaces. He received his PhD from the University of Nottingham. He is a member of Hong Kong Computer Society. Contact him at Hong Kong Baptist Univ., Computer Science Dept., 224 Waterloo Rd., Kowloon Tong, Hong Kong; kckwong@comp.hkbu.edu.hk.

Ka Keung Hui is a freelance software developer. His research interests are machine learning and human-computer interfaces. He is a graduate of Hong Kong Baptist University, where he received his BSc (Hons.) in applied computing and M.Phil. in computer science. Contact him at Hong Kong Baptist Univ., Computer Science Dept., 224 Waterloo Rd., Kowloon Tong, Hong Kong; kkhui@comp.hkbu.edu.hk.