Automated User Modeling for Intelligent Interface

Kenichi Yoshida and Hiroshi Motoda

Advanced Research Laboratory, Hitachi Ltd.
Hatoyama, Saitama, 350-03 Japan

The analysis of the user behavior is one important function of the intelligent user interface because, by analyzing the user behavior, it becomes possible to understand the user intention and release the user from tedious tasks which are often required to use a fast but low-level interface. The acquisition of the user behavior model is crucial. Most studies meant to realize an intelligent interface system only analyze superficial user behaviors, from which to automate the repetitions. Their user models tend to be simple and do not reproduce the behavior well enough. This paper presents a new framework that analyzes the computational processes activated by the user commands to build the user behavior model. An important feature of the proposed framework is the analysis of data dependency between the user commands. A user-adaptive interface system, *ClipBoard*, was developed to show the adequacy of this framework. It analyzes the I/O relationship between applications in the past task history, selects the next application, and creates scripts which enable complex task execution by a single command.

1. Introduction

As a task becomes more complex, the number of applications needed to perform the task increases and the process to use the computer system becomes more complex and difficult. A single data file may be processed by multiple applications. Correspondingly, a single task may be performed by a combination of multiple applications. The analysis of the user behavior is one important function of the intelligent user interface because, by analyzing the user behavior, it becomes possible to understand or at least conjecture the user intention and release the user from tedious tasks which are often required to use a fast but low-level interface.

Most mouse-based interface systems, such as Microsoft Windows and Apple Macintosh, have the *drag and drop* function to explicitly specify applications. These systems also have a function that specifies a default application for each file. While the users do not need to specify application programs during normal operation, they have to select an appropriate application to use the *drag and drop* function when the same file is used for more than one application. For example, *latex* is a well-known document formatter. Since its function is only to format a document, the *latex* user employs a separate editing system, such as *emacs*, to modify the document contents. The document processing task with *latex*, therefore, involves an interchangeable use of *emacs* and *latex*, and the use of the default does not help in this situation because both systems process the same file interchangeably.

Conventional interface systems also have a function that permits use of script language to program a complex task by a combination of simpler applications. However, the user needs some programming skills, which reduces user friendliness of the interface.

ClipBoard is a user-adaptive interface system for UNIX operating system that automatically selects an appropriate application to support complex tasks. It analyzes the I/O relationship between applications in the past task history, selects the next application, and creates scripts which enable complex task execution by a single command. The knowledge base to select an appropriate application is automatically generated by analysis of the I/O relationship between applications, and thus, no pre-specified knowledge base is necessary. The information necessary to generate scripts is also acquired during the operation of applications.

The use of the I/O relationship between applications increases the selection accuracy and facilitates the automatic script creation. Information on the data dependency between commands facilitates induction by providing an additional source of information. Experiment shows the evidence that this additional information gives a better user behavior model, and improves the selection accuracy.

2. Related work

The intelligent user interface is an important area of investigation. EAGER [2] is a HyperText system that observes user operations, finds repetitions, and offers to automate repetitive operations. It has a special mechanism to generalize loop counts and makes macro operations that perform repeated operations. The knowledge which automates the repetitive operations is automatically acquired. [4] analyzes the pattern of repetitions in the UNIX command histories and observes some regularities. [7] discusses a version of the emacs editor that also extracts repetitions from user operations to make the editing task more efficient. There are some work that used machine learning techniques. [3,5] used a decision tree learning method and [6] examined a K-nearest neighbor method. These efforts resulted in various applications: *e.g.*, a meeting scheduling agent, an electronic mail agent, and a form filling system.

These studies analyze only user operation sequences, and extract repetitions from therein. The applications were carefully selected and no other source of knowledge was required. However, this success can't be generalized to other types of interface systems such as command prediction and script generation. We need to use other source of knowledge as well as sequence information. The most typical approach would be a knowledge-based approach. Both APU [1] which is the UNIX shell-scripts programming system, and Gold [8] which is a business charts editor, take this approach. The knowledge must be carefully hand-coded in these systems.

The aim of this study is to build a user-adaptive intelligent interface by using, in addition to the sequence information, other source of knowledge which can be automatically collected during the user operation, i.e. the I/O relationship between applications. Since this information has rather complicated structure, most of the conventional inductive learning techniques is not adequate to be applied. We use graph-based induction method [10] to generate the user behavior model.

3. Overview

Figure 1 shows the system configuration of *ClipBoard*. It has three main components: a mouse-based application controller, a subgraph extraction program, and an I/O recorder. The I/O recorder is part of the operating system and records all the I/O operations of each application program. Suppose a user is making a document using a *latex* document formatter according to the following process:

1. Make a document source by an *emacs* editor.
2. Use the *latex* document formatter to check the document's bibliography.
3. Use the bibliography processor *bibtex* to make a bibliography data base *paper.bbl*.
4. Use the *latex* document formatter to access the bibliography data base.
5. Use the *latex* document formatter to make a final copy.
6. Preview the final copy by the *xdvi* previewer.

Figure 2 shows the information recorded by the I/O recorder in a simplified form.

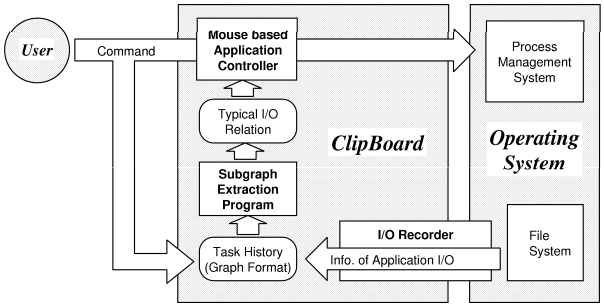


Figure 1. Overview of system configuration

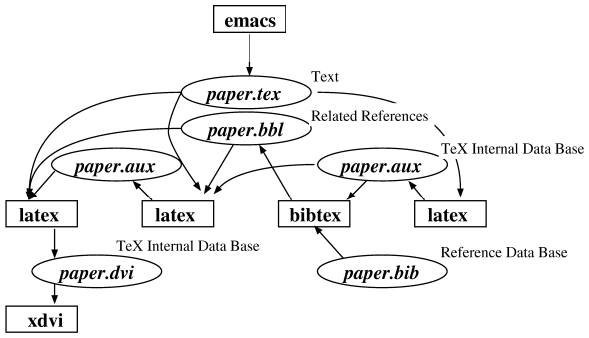
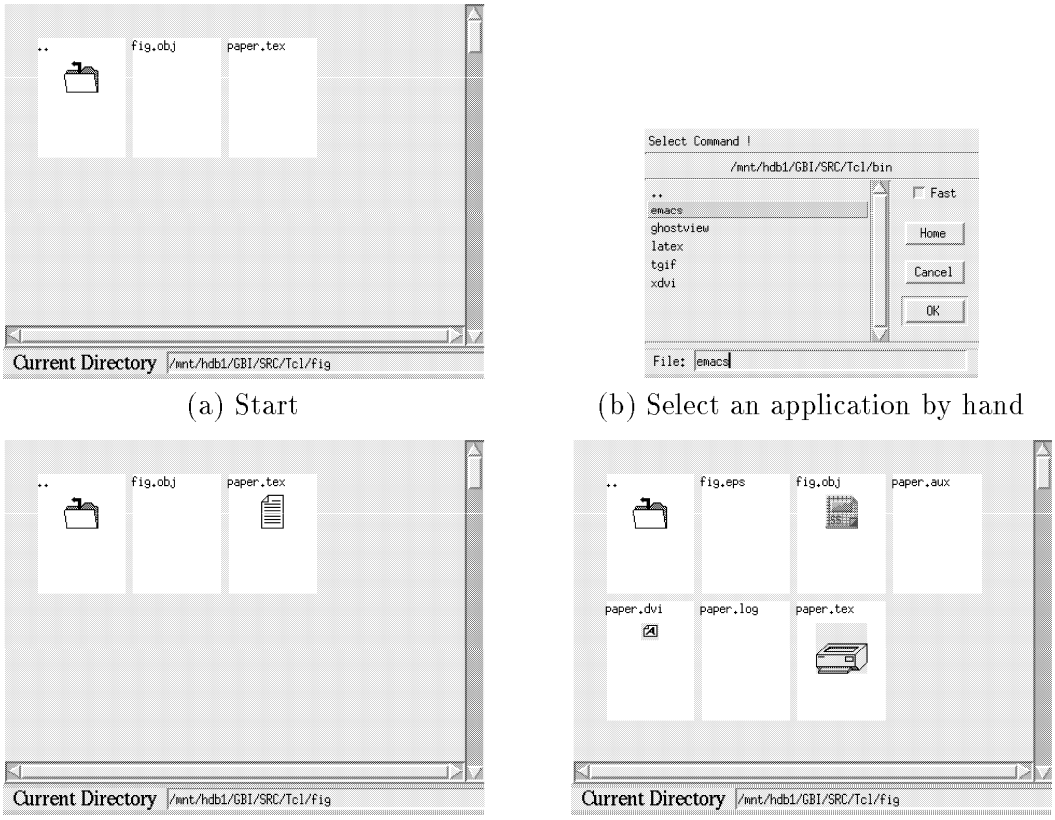


Figure 2. I/O relationship between applications

We use a directed graph as the representation language of this information. In Figure 2, the leftmost *latex* node takes *paper.tex*, *paper.bbl* and *paper.aux* as inputs. The edges are linked via these inputs to other nodes: *emacs* (creator of *paper.tex*), *bibtex* (creator of *paper.bbl*), and *latex* (creator of *paper.aux*). We also record the application sequence, which is the main source of the information that the conventional induction programs can use. One edge has a role of recording the preceding applications (sequence information). This can be used to learn the user preference of the independent application sequence such as "first check mail, then read news" when the sequence information does not represent dependency. However, this edge has been omitted from Figure 2 to improve readability.

The information acquired by the I/O recorder is analyzed by the subgraph extraction program. Using graph-based induction method, this program extracts typical subgraphs from the input graph so that the extracted subgraphs represent typically used application patterns. The mouse-based application controller uses these extracted patterns to guide the selection of the next applications.

Figure 3 displays a simple document processing task with ClipBoard. When ClipBoard



(a) Start (b) Select an application by hand

(c) *ClipBoard* suggests *emacs* for *paper.tex* (d) *ClipBoard* suggests *latex* for *paper.tex*

Figure 3. A simple document processing task

starts without any information about applications, the screen only lists file names (Figure 3 (a)). At this stage, after selecting (*i.e.*, clicking the left mouse) the file to be processed, the dialogue box appears so that the user can specify the application (Figure 3 (b)). If the user specifies *emacs*, ClipBoard treats *emacs* as the default for the file with suffix *tex* (Figure 3 (c)). The user can override the ClipBoard's suggestion by means of the dialogue box (Figure 3 (b)) which can be invoked by the right mouse button.

During the task, ClipBoard analyzes the I/O relationship between applications and extracts typical relationships. Which application to suggest changes over the time according to changes in extracted relationships. If the result of induction over the past task history suggests another appropriate application, ClipBoard changes the icon on the screen for a new application as the default. For example, ClipBoard changed its suggestion for *paper.tex* to *latex* in Figure 3 (d). Note the change of icons from *a paper sheet* (Figure 3 (c)) to *a printer* (Figure 3 (d)).

In a typical document processing task with *latex*, ClipBoard suggests *emacs* and *latex* interchangeably as the applications for the file *paper.tex* to use next. As shown in Figure 4, the additional information, *i.e.*, information on the data dependency between the

user commands, can induce a better user behavior model, and improves the prediction accuracy. The results shown here is the average of about 100 commands over three months history of a single user. If we limit to the major commands, such as *latex* and *xdvi*, the accuracy goes up to 80 to 100%.

ClipBoard also makes scripts which enable the execution of complex task by a single command. Figure 5 shows an example of the generated scripts when a user repeatedly calls up *emacs*, *latex* and *xdvi*. Note that we also use graph-based induction method to create scripts. In other words, we use the notion of typically used application patterns to both command prediction and script generation. Thus, the typical application pattern is used as a model of user's behavior.

Method	LD	1-NN	CART	CLiP
Accuracy (%)	22.6	20.8	34.6	57.8

LD: Linear discrimination method
1-NN: 1 nearest neighbor method
CART: Decision tree learning method
CLiP: Proposed method

Figure 4. Selection accuracies

```
#! /bin/csh -f      # Document Processing Script.

emacs $1            # Edit Document.
                    # Suffix is assumed to be "tex".

latex $1            # Format Document.

                    # Preview Result on Screen.
xdvi $1:r.dvi       # Suffix is assumed to be "dvi".
```

Figure 5. Example of the generated script

4. Discussion

Although the graph-based induction method considers only the syntactical/statistical nature of the application patterns in the history, we could show that it can extract patterns which seem to have some important meaning (semantics) of the user task. The most important factor here is the criterion used to select the pattern. The graph-based induction method uses various techniques developed by the statisticians (e.g. gini index, cross validation etc.). Both the improvement of the selection accuracy and the adequacy of the generated scripts in the experiments suggest the usefulness of our approach.

We note that the techniques developed by the statisticians are one important factor which extracts semantics from syntactical information. However, the I/O information used by the analysis is also another important factor. In particular, we use file suffix information. In UNIX and similar operating systems, the file suffix tends to represent the contents of the file. For example, a file with suffix *tex* is a *tex* (or *latex*) document file, and a file with suffix *c* is a C program source file. Thus, file suffix information also contributes to the extraction of the promising patterns.

[9] notes the importance of the *context* for the interface system. They use high level information such as *word* or *line* to express the context in the word processor. The file suffix information used in our study also provides the context information in a similar manner. However, we have not yet made a thorough investigation of the available information to capture the context, which remains to be an important future research issue.

5. Conclusion

The typical approach to building an intelligent interface observes the user's behaviors and tries to imitate them. We limited the scope of the user behavior to a sequence of task (e.g. editing, formating, viewing, etc.) execution using plural application programs, and assumed that if a model can predict what the user is supposed to do in the same situation, the model represents the user model. Most conventional studies analyze only superficial repetitive user behavior, from which to automate the repetition and completion. However, these methods lack a framework for using the information about the relationships between the applications which is important both to improve the accuracy and to create command scripts. In this paper, a new framework that is based on graph-based induction has been introduced to overcome this limitation.

The I/O relationship between applications is the main information source of the analysis. Although the proposed method considers only the statistical/syntactical nature of the user behaviors, it can extract patterns which seems to have some important meaning (semantics) for the user task. The generalized idea is the use of data dependency between sub-tasks to analyze the relationship between them. This idea seems to be useful in designing an interface system for more complex applications such as data base systems. However, evaluation of such systems remains to be a future research issue.

REFERENCES

1. S. Bhansali and M. T. Harandi. Synthesis of unix programs using derivational analogy. *Machine Learning*, 10:7–55, 1993.
2. A. Cypher. EAGER: Programming Repetitive Tasks by Example. In *CHI'91*, pages 33–39, 1991.
3. L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. A Personal Learning Apprentice. In *AAAI-92*, pages 96–103, 1992.
4. S. Greenberg and I. H. Witten. How Users Repeat Their Actions on Computers: Principles for Design of HISTORY Mechanisms. In *CHI'88*, pages 171–178, 1988.
5. L. A. Hermens and J. C. Schlimmer. A Machine-learning Apprentice for the Completion of Repetitive Forms. In *Proc of the Ninth Conf. on Artificial Intelligence for Applications*, pages 164–170, 1993.
6. P. Maes and R. Kozierok. Learning Interface Agents. In *AAAI-93*, pages 459–465, 1993.
7. T. Masui and K. Nakayama. Repeat and Predict - Two Keys to Efficient Text Editing. In *CHI'94*, pages 1178–123, 1994.
8. B. Myers, J. Goldstein, and M. Goldberg. Creating Charts by Demonstration. In *CHI'94*, pages 106–111, 1994.
9. P. P. Piernot. The AIDE Project: An Application-Independent Demonstrational Environment. In A. Cypher, editor, *Watch What I Do: Programming By Demonstration*, pages 387–405. MIT Press, 1993.
10. K. Yoshida, H. Motoda, and N. Indurkhya. Graph-based Induction as a Unified Learning Framework. *Applied Intelligence*, 4:297–328, 1994.