Picture Script: Visualizing Graphical User Interface Commands for Recycling

Sasakura, Mariko; Iwata, Kenichi; and Yamasaki, Susumu

Abstract—*We propose a system that visualizes and records a command that is used in the graphical user interface. The basic idea is that the interface visualizes a movement of a mouse or pen and remains it on a screen. We represent our user interface as a kind of visual language, and construct a mechanism for the analysis of the language. In this paper, we show a small application using the user interface that performs the file transfer protocol (FTP). Recording a command used in the graphical user interface will be able to extend the interface as recycling a command and performing batch operations.*

Index Terms—*System Software, Application Software, Visual Language, Graphical User Interface*

1. Introduction

RECENTLY, the mainstream of the user interface for personal computers is the graphical user interface (GUI). The advantages of the GUI are expected:

- Users don't have need to remember commands or options because they are shown as icons or listed in menus.
- Operations, that are clicking or drugging by a mouse, are intuitive and easy to learn.

These advantages are not only for beginners but heavy users also. Major integrated development environment products, which are used to build application programs, are working under GUI. They are used by programmers widely, who are not beginners but heavy users.

The GUI has a disadvantage:

- When we have to repeat similar operations again and again, for example, when we manage plenty of files, the work is too boring and takes a lot of time.

We propose a user interface that allows a user to “recycle” previous operations of GUI commands. In this paper, we use the word “recycle” of which meaning is that we can reuse and modify the previous operations to make new operations.

In the character-based user interface (CUI), like the UNIX shell, the system can remember a history of commands and represent the executed command to a user in order to modify and recycle it. In the GUI, we have functions of undo and redo that are similar to the history of commands in the CUI. However, by the undo and redo functions, the system does not remember a history of commands or operations but just states. We can move to the previous states, but we cannot recycle previous operations that we executed.

Recycling operations is useful for a heavy user of the GUI. When we rename plenty of files or organize files in the appropriate folders, we do similar operations, clicking and/or drugging, a dozen times. If we can recycle the previous operations, these works may be easier.

To establish a new graphical interface in which we can recycle operations, we have the following problems:

- What operations are recycled?
- How do we recycle operations?

Our solutions are as follows.

- We formalize movements of a mouse/pen as a visual language.
- We remain a trace of a mouse/pen movement on a screen for further recycling.

In this paper, we mention *picture script interface* in which we can recycle a GUI operation, and *picture script* which is a visual language used in the picture script interface. We also show a small application of the picture script interface which represents the file transfer protocol (FTP) commands.

2. Problems And Related Works

In this section, we discuss the details of problems and show the solutions we proposed. Then we mention related works.

2.1 The Details of Problems and Solutions

We propose a new user interface which allows a user to recycle previously executed operations. The basic idea is visualizing a movement of mouse/pen and remaining it on a screen. In this subsection, we describe the details of problems.

There are problems:

(1) What operations of the graphical user interface are recycled?

(2) How do we recycle operations of the graphical user interface?

A general GUI shows icons for resources and operates them by a mouse/pen interface. It is hard

Manuscript received 28 July, 2005; revised 6 January, 2006

Authors are with Department of Computer Science, Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-Naka, Okayama, JAPAN
(E-mail: sasakura@momo.it.okayama-u.ac.jp).

This research is supported by the Okawa Foundation。

to recycle operations in the GUI because they are disappeared. Therefore for the first problem:

- We try to leave the trace of movements of the mouse/pen so that we can show the previous movements in the screen. Then, we can recycle them.
- We formulate a visual language in order to specify what operations are recycled. It gives the computer scientific basis for our approach.

For the second problem:

- We construct a mechanism of analysis of the visual language.
- We design no button or menu for the picture script interface because the interface is intended to be attached to the existent graphical user interfaces.

2.2 Related Works

2.2.1 Existing Solutions

We mention the researches which allow a user to recycle graphical operations and compose macro commands by them. Borg proposes a visual programming environment for UNIX shell [3]. It is a kind of iconic command language and can specify a script. Shigesada at el. propose an icon based user interface [27] which is worked on TRON [33]. It aims to perform batch operations on the graphical user interface. Modugno and Myers develop a visual shell which allows a non-programmer to construct a macro program [20]. The motivation of the researches is similar to ours, but they use icons while we use traces of movements of a mouse/pen. We think that recycling movements of mouse/pen is more convenient than learning new icons.

About grammars and parsers about visual languages, Chok and Marriott [6], and Helm and Marriott [11] propose a general one. We define a simple visual language which is based on the spatial relation between figures.

2.2.2 Related Visual Languages Works

Our research is regarded as one of the visual languages. The studies on visual languages started to provide programming languages that were easy to learn and use [5], [19], [21].

It is popular for visual languages or notations to be used for helping users understand programming languages that is difficult to understand. For example, a formal specification language LOTOS has its graphic form G-LOTOS [15]. Agusti at el. propose a visual logic programming language [1] to provide the graphical intuitions for understanding first order logic. Sasakura at el. propose a system that visualizes processes of reasoning by logic program [26] and the application system of it for a product managing system [16], [34]. There are many researches for providing languages with visual forms [2], [8], [12], [13], [17], [25],.

On the other hands, there are some researches that are apart from the concept of traditional programming languages but are suitable for simulation, especially by children. Viscuit [10], KidSim [7], and AgentSheet [23] are typical works.

Recently, the works on domain-specific visual languages are active [4], [30]. Visual languages for web services are also popular theme [22], [28].

The history of studies on visual languages shows that they do not take the place of the traditional textual programming language. They should have other roles by making good use of their advantage which is users can easily understand visual notations.

3. *The Picture Script Interface*

The condition for applying the picture script interface to an application is that the application needs mouse/pen movements. Especially in applications which have frequent movements of a mouse/pen, the interface should be useful: for example, managing plenty of files. In this paper, we show how we implement its interface with an example application.

In this section, we show the architecture of the picture script interface.

3.1 System Overview

The overview of the picture script interface system is shown in Figure 1. The system consists of two parts: the picture element analysis and the picture rule analysis.

The tasks of the interface are listed:

1. It recognizes a drawn figure. This task is done in the picture element analysis.
2. It interprets the figure according to rules which represent the correspondence between figures and commands for some application. This task is done in the picture rule analysis.
3. It sends the commands for an application, which are specified by the figures.

The first task corresponds to lexical analysis in compilers, the second to syntax analysis, and the third to code generation, respectively.

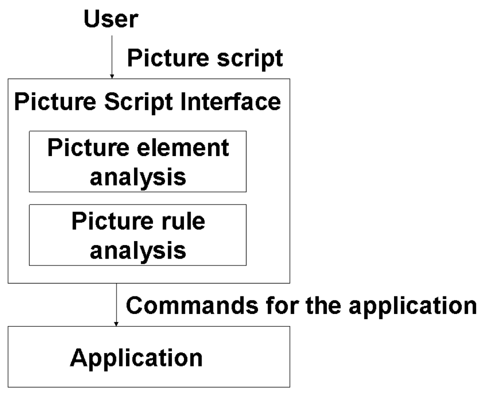

Figure 1 System overview.

3.2 Picture Element Analysis

The picture element analysis part is responsible to the recognition a drawn figure. It is to analyze a figure and recognize it as *elements*. An element is a line, an arrow or a string which is a sequence of characters.

At first, a user draws a figure on a canvas we provide. We call the all things written in a canvas as a figure. The picture element analysis part divides a figure into elements which corresponds to tokens in compilers.

3.3 Picture Rule Analysis

The picture rule analysis part is responsible to the interpretation of the figure and the translation of it to commands for an application. It translates a sequence of elements that satisfies a given syntax rule to commands for an application. We call the sequence of elements as a *picture*.

After recognizing elements in a figure, we check whether connections of elements are the same as one of given rules. Rules can be given by users. We mention what rules we make for an example application in section 4.2.2, and how we check a connection of elements in section 4.3.2.

4. THE PICTURE SCRIPT LANGUAGE AND ITS INTERPRETATION

The picture script is a kind of visual languages. Its target code is a set of commands for some application. Elements are regards of tokens of the language and rules specify its syntax. In this section, we describe a picture script language with a simple application in a formal way, and show how the language is interpreted.

4.1 A Simple Example

To explain the picture script language, we give a simple example that is an file transfer protocol(FTP) client. Since the aim of the example is demonstration of the picture script, our FTP client has limited ability. It knows only the following FTP commands:

- open a host
- get a file
- put a file
- close

4.2 Syntax

4.2.1 Elements

Elements are the elemental unit of the picture script. They may be regarded as tokens of it. In this application, we use the following four types of elements:

- a line
- a string which is a sequence of alphabet characters
- a left arrow and
- a right arrow

An arrow consists of a line and an arrowhead. A left arrow has its arrowhead on the left side of the line, and a right arrow has on the right side.

We design the system is robust in recognizing elements because the purpose of the system is not to draw beautiful or precise figures but to provide a convenient user interface. In this system, if more than one shorter line exists near of the start or end point of a longer line, we regard them as an arrow. Figure 2 shows examples of arrows in our system.

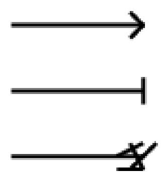

Figure 2 Arrows. Any of the elements shown in this figure is recognized as an arrow in the system.

4.2.2 Rules

In this subsection, we define rules for interpreting a picture to an FTP command in formal way. A picture is a sequence of elements that is defined by the following definition.

Definition 1 *A picture is recursively defined to be:*

(1) an element (a line, an arrow or a string), or
(2) **near**(A, B), **nearCenter**(A, B) or **overlap**(A, B), where A and B are pictures.

near, **nearCenter** and **overlap** are binary operators. The meanings of the pictures are represented by the operators:

- **near** (A, B) means that A is close to B and A places the left of B,
- **nearCenter** (A, B) means that the center of A is close to the center of B, and
- **overlap** (A, B) means that A overlaps B.

At most one of the operations is to hold between any two pictures.

Definition 2 *A rule is written by the following form.*

A command ::= a picture

where "::=" represents the translation from a picture to a command. A rule means if the picture is found in a figure, the command is sent to the application.

By these definitions, we represent FTP commands of the system as the following:

Definition 3 *The rules that we use for the FTP application are listed below.*

- open A ::= **near** (HERE, **near** (Line, A))
- get F ::= **near** (HERE, **nearCenter** (F, **near** (LeftArrow, A)))
- put F ::= **near** (HERE, **nearCenter** (F, **near** (RightArrow, A)))
- close ::= **near** (HERE, **overlap** (Line, **overlap** (Line, **near** (Line, A))))
- close ::= **near** (HERE, **overlap** (Line, **overlap** (Line, **near** (Arrow, A))))

HERE is a keyword that represents the machine adopted by the user. “Line” means a line. “A” and “F” represent strings. “LeftArrow” means a left arrow and “RightArrow” means a right arrow. “Arrow” means both of left and right arrows.

If a picture occurs in the right-hand side of the rule, we say the picture is interpreted by the rule.

Proposition 1 *A picture can be interpreted by at most one rule of Definition 3.*

Proof.

(1) *Any rule is not the same as another rule, and*

(2) *any rule is not implied by other rules*.

Therefore, a picture cannot be satisfied by the interpretation of two rules.

4.3 Syntax Analysis

4.3.1 Element Analysis

Recognition of elements is simple. In our system, we can use a mouse/pen and a keyboard. When we use a keyboard, an input element must be a string. By a mouse/pen, an input element may be a polyline. If a polyline has only two points which are the start and end point, it may be a line. Otherwise it may be curved line. Then we do not need any button to input an element.

An arrow is a set of lines. Therefore we must analyze which set of lines forms an arrow. An arrow consists of an arrow body and an arrowhead. An arrow body is a line. An arrowhead is more than one line. The line for an arrow body must be longer than any line for the arrowhead. Then, an arrow decision algorithm when a line L is drawn is the following.

Algorithm 1

```
if here is an arrow A and the line L is
near by the start or end point of A and
the body of A is longer than L then
     L is an arrowhead of A
else if there is a line B and the line
L is near by the start or end point of
B and B is longer than L then
     A and B construct an arrow of
     which body is A and arrowhead is B
```

4.3.2 Rule Analysis

The rule analysis checks whether a set of elements satisfies any given rule. This consists of two steps.

- Step1: check the type of elements: a line, a string or an arrow, and
- Step2: check whether connections of elements satisfy a rule.

All elements (includes the result of binary operators) have the right point and the left point. The right point is the most right-up point of an element. The left point is the most left-up point.

Then, an algorithm for rule analysis is the following:

Algorithm 2

```
Put all elements into an array A in the
order of the right point.
  for all rules
   begin
   for i = 0
       to the number of elements
     begin
     for j=i to the number of
            elements of a rule
      begin
      if the types of elements from
         A[j] match the types of
         elements of the rule
      then
        begin
        if all relations between the
           elements from A[j] satisfy
           the rule
        then
           send an FTP command to the
           FTP client
        end
      end
     end
    end
```

This check algorithm is invoked whenever an element is added. In order to avoid sending the same command more than once, the system records the elements that are used to send a command.

5. Implementation of the Picture Script Interface

In this section, we mention implementation issues of the picture script interface. We design a file transfer protocol (FTP) client as an example of the interface. The FTP client is implemented as an example of the application which is indicated in Figure 1.

5.1 An FTP Client Implementation

An FTP command that is the output of the picture rule analysis should be sent to an appropriate FTP server. We design an FTP client which has an interface to the picture script interface:

- Our FTP client receives a message from the picture script interface and sends it to an appropriate FTP server. And it receives a message from an FTP server and sends back it to the picture script interface.

The feature of the picture script interface is that it needs no button and menu on the canvas. A user just draws a figure by a mouse or pen, or types a keyboard to input alphabet characters. Therefore our FTP client has no button and menu in its user interface.

The FTP client works according to the following cycle.

1. A user inputs diagram by a mouse or pen, or characters by a keyboard.

2. The system recognizes the input and checks rules.
3. If there is a proper rule, the system sends an appropriate message to the FTP client.
4. Return to 1.

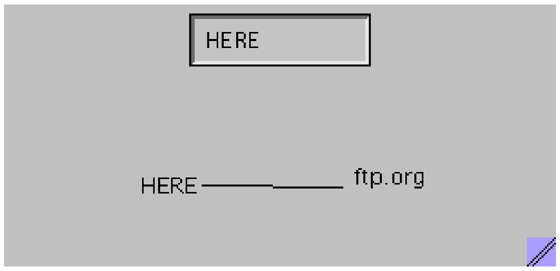

Figure 3 The “open” command. The line between “HERE” and “ftp.org” specifies to open a connection to ftp.org.

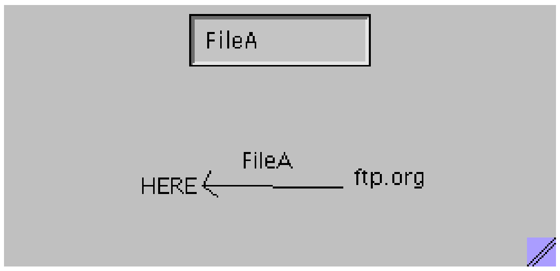

Figure 4 The "get" command. The “get” command is made by recycling the line for the “open” command.

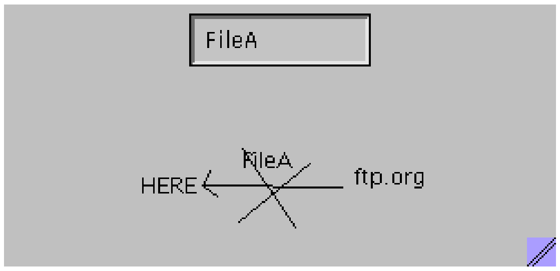

Figure 5 The “close” command. The “close” command is made by recycling the line for the “open” command.

Figures 3, 4 and 5 show the user interface of the system. In Figure 3, we can see a line and two strings which are placed in the left side and right side of the line. It means opening a connection between ftp.org and the machine what a user is using. Figure 4 shows an arrow and two strings. This figure is constructed by adding an arrowhead to Figure 3. It means getting the FileA from ftp.org. Figure 5 has the same elements with Figure 4 except two lines which are crossed on the arrow. It means cutting the connection. In these figure, we can send an FTP commands by recycling the previous elements.

An element can be deleted by drugging it to out of the canvas of the picture script interface. Copying an element and scaling an element are not implemented in this system.

By the inputs shown in Figures 3, 4 and 5, the picture script system sends (a) open (b) get file and (c) to our FTP client. Our FTP client is implemented by Perl. It can receive messages from the picture script system and send them to FTP servers. And also it can receive messages from FTP server and an appropriate message to the picture script system.

5.2 Discussion

5.2.1 Advantages and Limitations of the Picture Script Interface

The picture script is a kind of visual languages that translates a picture to any kinds of commands. Therefore, in principle, the picture script interface can be applied to any GUI application.

The advantages of the picture script interface are the following points.

- Recycling: a picture recorded in the system can be recycled. If we want to send the same command as previously, we may click the picture.
- Extendibility: if we give other set of rules to the picture script interface, it works according to the new rules.
- Convenience: the picture script interface could be implemented as a front end processor so that it interprets a picture to any application.

The interface is limited:

- We can use only lines, arrows, and characters. We cannot use a rectangle, a circle, a curved line.
- The picture script interface assumes that a figure must be analyzed in horizontal way.

To raise the limitations:

- For recognizing more elements, we can use other researches on a freehand interface [14], [24].
- We may analyze a figure by not only from the left to the right but also any direction.

5.2.2 Applications

The picture script interface can be applied to any GUI application. In this subsection, we list applications in which the picture script interface is useful.

- When we want to process many e-mails by a SPAM filter program.
- When we want to house many files into many folders.
- When we develop a program on an integrated development environment.

The picture script can visualize the history of executed commands and recycle them. This ability allows us to make a macro, in other words, a batch operation, for any GUI. Current GUIs almost do not have the ability of defined macros by a user. The picture script can provide a convenient interface to a user.

We design the picture script for graphical user interfaces. However, it can be used for other purposes. For example, it may be useful to write a specification of a system, especially a concurrent system, with a specification language [9], [18], [29], [31], [32].

6. Concluding Remarks

We propose the picture script interface that can recycle mouse/pen operations for any application. The basic idea is that visualizing a movement of mouse/pen and remaining it on a screen. We establish the picture script as a visual language so that we show its syntax and analysis mechanism in this paper. We also show an example application that performs the file transfer protocol. The picture script interface achieves:

- Recording pictures for further recycling
- Translating from a picture to commands for any application consistently

References

[1] Agusti, J., Puigsegur, J. and Robertson, D., “A visual syntax for logic and logic programming”, *J. of Visual Languages and Computing*, vol.9. no.4, pp.399-427, 1998.

[2] Bonhomme, C., Aufaure, M.-A. and Trépied, C., “Metaphors for visual querying spatio-temporal databases”, *Proc. of 4th International Conf., VISUAL 2000, (Lecture Notes in Computer Science vol.1929)*, pp.140-153, 2000.

[3] Borg, K., “IShell: a visual UNIX shell”, *CHI '90*, pp.201-207, 1990.

[4] Bottoni,P., De Marsico, M., Di Tommaso, P., Levialdi, S. and Ventriglia, D., “Definition of visual processes in a language for expressing transitions”, *J. of Visual Languages & Computing*, vol.15, no. 3-4, pp.211-242, 2004.

[5] Burnett, M. M. and Baker, M. J., “A classification system for visual programming languages”, *J. of Visual Languages and Computing*, vol. 5, no.3, pp. 287-300, 1994.

[6] Chok, S.-S. and Marriott K., “Automatic construction of user interfaces from constraint multiset grammars”, *11th IEEE International Symposium on Visual Languages*, pp.242-249, 1995.

[7] Cyper, A. and Smith, D.C., ”KidSim: end user programming of simulations”, *ACM Conf. companion on Human factors in computing Systems*, pp.35-36, 1995.

[8] Erwig, M. and Schneider, M., ”A visual language for the evolution of spatial relationships and its translation into a spatio-temporal calculus”, *J. of Visual Languages and Computing*, vol. 14, no.2, pp.181-211, 2003.

[9] Galloway, A. J. and Stoddard, W. J., “An operational semantics for ZCCS”, *First IEEE International Conf. on Formal Engineering Methods*, pp.272-282, 1997.

[10] Harada, Y. and Potter, R., “Fuzzy rewriting - soft program semantics for children –“, *IEEE Symposium on Human Centric Computing Languages and Environments*, pp.39-46, 2003.

[11] Helm, R., Marriott, K. and Odersky, M. “Building visual language parsers”, *ACM Conf. Human Factors in Computing*, pp.105-112, 1991.

[12] Hewagamage, K. P. and Hirakawa, M., ”An interactive visual language for spatiotemporal patterns”, *J. of Visual Languages and Computing*, vol.12, no.3, pp.325-349, 2001.

[13] Hirakawa, M., Tanaka, M. and Ichikawa, T., ”An iconic programming system, HI-VISUAL”, *IEEE Trans. on Software Engineering*, vol.16, no.10, pp.1178-1184, 1990.

[14] Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H., “Interactive beatification: a technique for rapid geometric design”, *Proc. of the 10th annual ACM symposium on User interface software and technology*, pp.105-114, 1997.

[15] ISO,”G-LOTOS: a graphic syntax for LOTOS”, *ISO/IEC JTC 1/SC 21 N 4228*, 1989.

[16] Iwata, K., Sasakura, M. and Yamasaki, S. "Visualization for management of electronics product composition", *Proc. of Ninth International Conf. on Information Visualization (IV05)*, pp.194-199, 2005.

[17] Koike, H., Takada, T. and Masui, T., “VisuaLinda: a framework for visualizing parallel Linda programs”, *Proc. of the 1997 IEEE Symposium on Visual Languages*, pp.176-182, 1997.

[18] Mahony, B. and Dong, J.-S., “Timed communication Object Z”, *IEEE Trans. on Software Engineering*, vol. 26, no. 2, pp.150-177, 2000.

[19] Marriott, K. and Meyer, B., *Visual language theory*, Springer, 1998.

[20] Modugno, F. and Myers, B. A., “Visual programming in a visual shell - a unified approach”, *J. of Visual Languages and Computing*, vol. 8, no.5-6, pp.491-522, 1997.

[21] Myers, B.A., “Taxonomies of visual programming and program visualization”, *J. of Visual Languages and Computing*, vol.1, no.1, pp.97-123, 1990.

[22] Pautasso, C. and Alonzo, G., “The JOpera visual composition language”, *J. of Visual Languages and Computing*, vol. 16, no.1-2, pp.119-152, 2005.

[23] Repenning, A. and Ambach, J., “Tactile programming: a unified manipulation paradigm supporting program comprehension, composition and sharing”, *1996 IEEE Symposium on Visual Languages*, p.102, 1996.

[24] Saga S.,”A freehand interface for computer aided drawing systems based on the fuzzy spline curve identifier”, *Proc. of 1995 IEEE International Conf. on Systems, Man and Cybernetics*, pp.2754-2759, 1995.

[25] Sasakura, M., Joe, K., Kunieda, Y. and Araki, K., "NaraView: an interactive 3D visualization system for parallelization of programs", *International J. of Parallel Programming*, vol.27, no.2, pp.111-129, 1999.

[26] Sasakura, M. and Yamasaki, S. "A three-dimensional visualization of communications in distributed program environments", *Proc. of Seventh International Conf. on Information Visualization (IV03)*, pp.132-137, 2003.

[27] Shigesada, Y., Koshizuka, N. and Sakamura, K., “Design of VACL: a visual script language system which controls and extends applications on a graphical user interface environment”, *13th TRON Project International Symposium/TEPS '96*, pp.104-14, 1996.

[28] Shimomura, T., “Visual design and programming for Web applications”, *J. of Visual Languages and Computing,* vol. 16, pp.213-230, 2005.

[29] Smith, G., *The Object-Z specification language*, Kluwer Academic Publishers, 2000.

[30] Sprinkle, J. and Karsai, G., “A domain-specific visual language for domain model evolution”, *J. of Visual Languages and Computing*, vol.15, no. 3-4, pp.291-307, 2004.

[31] Spivey, J.M., *The Z notation*, Prentice Hall, 1992.

[32] Taguchi, K., Dong, J.-S. and Ciobanu, G., “Relating pi-calculus to Object-Z”, *Ninth IEEE International Conf. on Engineering Complex Computer Systems*, pp.97-106, 2004.

[33] TRON, http;//www.tron.org/index-e.html.

[34] Yamasaki, S., Iwata, K. and Sasakura, M., "Reasoning procedure and implementation for logic programs as managing schemes to extract demand", *IPSI Trans. on Advanced Research*, vol.1, no.1, pp.83-90, 2005.

Mariko Sasakura received Eng.D. in information science from Kyushu University in 2000. She is an assistant professor of Okayama University.

Kenichi Iwata is a Ph.D. candidate at Graduate School of Natural Science and Technology, Okayama University.

Susumu Yamasaki received Eng.D. in information science from Kyoto University in 1980. In 1985/86, he was a visiting fellow at Dept. of Computer Science, University of Warwick, U.K. In 1987, he joined Okayama University as a professor.