Proceedings of 2nd International Conference on Information Technology Based Higher Education and Training, Kumamoto, Japan
July 4-6, 2001

Programming Education on an Interactive Electronic Whiteboard

Masaki Nakagawa, Taro Ohara, Jin Kanda Hirokazu Bandoh and Naoki Kato
Department of Computer, Information and Communication Sciences
Tokyo University of Agriculture and Technology
nakagawa@cc.tuat.ac.jp

Abstract

This paper describes a new paradigm of programming education using a large electronic whiteboard that combines the merits of classroom lectures using a black/white board and those of computer processing. Using this system, a teacher can write a program on the board, explain it, make the system recognize it and run the program in front of the class while keeping the attention of the students focused on the board. The system allows input data to be entered by writing them on the board.

1. Introduction

So far, we have been teaching programming on a white/black board and telling students to try the programs that we have explained after the class. Next week, their answer is often that the programs have not worked. The reason is quite simple. Programs that we write on a white/black board are not always syntactically perfect and moreover students often misread them or miscopy them in their notebooks.

Then, the students ask us to lecture in a computer room. Now, they can try programming immediately. But, the problem is that they focus their attention on their PCs and do not pay attention to our explanations. It is very difficult to present intrinsic materials in a computer room.

On the other hand, the advantage with the white/black board is: teachers can write or draw what they want to express most easily using a chalk; teachers can gather the attention of students to their writing; teachers and students are familiar with it; teachers can have the control over class learning; and teachers can present new materials while confirming understanding of students by watching their faces. But, we cannot show the execution of programs in front of them.

Although practice on a PC is essential for students to learn programming for themselves, to computerize the white/black board and provide it with the capability of showing program execution brings about new potential for programming education.

We decided to develop a system to combine the merit of the white/black board and that of computer processing as a part of pen interface research. The system may open the way for new applications other than programming education. We call our system *IdeaBoard* (Interactive, Dynamic, Electronic Assistant Board) and show its appearance in Figure 1.

We have already presented hardware, new user interfaces and some educational applications in user interface communities [7-10]. In this paper, come back to our original motivation to develop *IdeaBoard* and describe the updated version of the programming education system.

2. User interface design

The design guidelines of the programming education system are based on those of the user interface with *IdeaBoard*. In this section, we summarize them since they are the basis for designing educational applications.

2.1. UI for a large surface with markers

If we just expand the usual GUI for the desktop machine to the board size, problems occur in its usability. A teacher has to move from side to side, stretch hands from button to button, which not only makes the teacher practice gymnastics but also often hides the board from

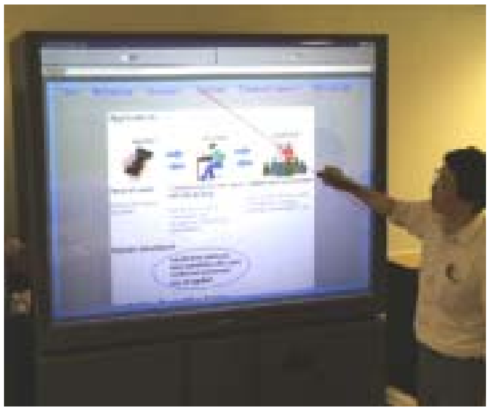


Figure 1. ***IdeaBoard*** **for Web browsing.**

the students. Often, operations too minute are also required which are difficult for the hand of a standing person. Moreover, double tapping with a marker is not as easy as with a mouse. Concurrent use of markers and a keyboard is far more difficult than that of a mouse and a keyboard because their manipulation scales are so different.

In order to design the interfaces, the size of the board, its vertical position, operability by markers, body actions by a teacher as well as the consistency with the desktop GUI must be taken into consideration.

2.2. As few gestures as possible

Gestures have been an attraction of pen interfaces from small size PDAs, note-size Pen-PCs [1] to large board systems like LiveBoard [2]. A gesture (pen gesture) can specify the kind of command and its target by a single pen action, but their employment without careful considerations may cause problems.

Gestures have simple shapes but simple shapes are hard for machines to recognize. There is little context to augment gesture recognition. Moreover, misrecognition of gestures or forgetting to set an appropriate mode before inputting gestures can yield an abrupt and unexpected result so that it not only interrupts lectures or presentations very badly but also make the user afraid to use gestures.

On large board systems, the gesture-command approach has been employed to solve the above-mentioned problems caused by a simple expansion of the desktop GUI to the board size [11] but we think that we should try to enhance the desktop GUI to make it suitable for large board systems without depending too much on gestures.

We think that pattern recognition can be more usefully employed for contents rather than for commands.

2.3. Extension of the desktop GUI

We are now very familiar with the desktop GUI. It has apparent advantages against the old style of user interfaces as well as being refined to be accepted from a majority of users.

Even if a simple expansion of the desktop GUI to the board size is not usable, to employ or extend its styles or elements in a form that they are consistent with the desktop while enhancing the usability of the board system can still be effective.

2.4. Design guidelines of the user interface

In the process of considering the user interface for large boards with markers and reviewing the desktop GUI, we have set the following guidelines:

(1) Operability from arbitrary standing position of the user
The teacher must be able to operate the board without having to cross the surface or stretch hands from side to side or from edge to edge.
(2) Easy operability with a single marker
This is to ensure that the teacher can operate the board with a single marker without needing other markers, keyboard, mouse, etc.
(3) Operability by body size movement
Too large a movement, required for a user to operate the board, is hard and ends up hiding the board, but too minute a movement is also difficult for the standing user. Body size movement with direct pointing and manipulation is not only natural to the user but also appealing to the audience and it navigates their focus of attention.
(4) Natural extension of the desktop GUI
This guideline ensures consistency with the desktop environment. Even when the styles and components of the desktop GUI must be modified based on the above considerations, the modification should be very natural so that users do not feel a big difference.
(5) Simplicity of displayed contents
This allows the teacher to operate the board without confusion and the students can understand the contents easily.
(6) More space for contents while less space for control
This is to make sure that the surface of the board is utilized for education as much as possible; buttons, menus, etc. should not hide the contents unless absolutely necessary.
(7) Smooth movement of displayed objects
Smooth and continuous movement of displayed objects seems more important for the shared large display so that the audience can keep their focus of attention without being annoyed by sudden changes of contents.

3. Design philosophy of educational software

Conventional CAI systems teach students instead of teachers. This is mainly designed for self-learning but not for IT-empowered lectures. On an electronic whiteboard, it is the teacher who teaches students. Therefore, we need to establish the design philosophy for this type of education software on the electronic whiteboard. Since several manufacturers are now producing electronic whiteboards, it seem important to establish the design philosophy and enrich educational applications according to the philosophy.

We set up the following design philosophy:

- Software does not explain. It is the role of teachers.
- Software provides IT-supported pieces of materials for the teachers to make a lecture.

Teachers make stories of lectures. They should be predefined in software.

4. Previous system

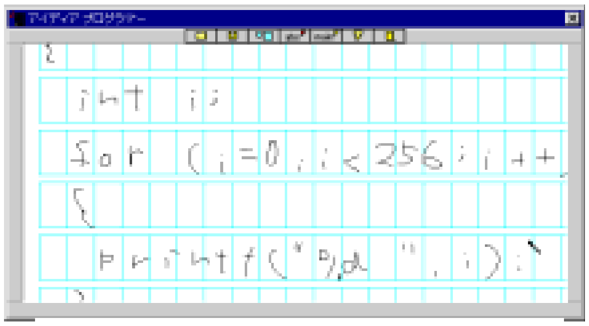
4.1. Overview

According to the guidelines, we made an initial version of the programming education system. A teacher can write a program in the C language into lines of character input frames as shown in Figure 2. These input frames are employed to facilitate handwritten program recognition. They are not hard restriction for the teacher to write a program unlike writing usual sentences into character frames one by one. The teacher can also load programs from files and save them back. Then, the teacher can direct the system to recognize a handwritten program as shown in Figure 3 and execute it as shown in Figure 4 by tapping menu buttons shown center in the upper or lower screen frame. The place to show the menu can be switched.

4.2. Screen scroll

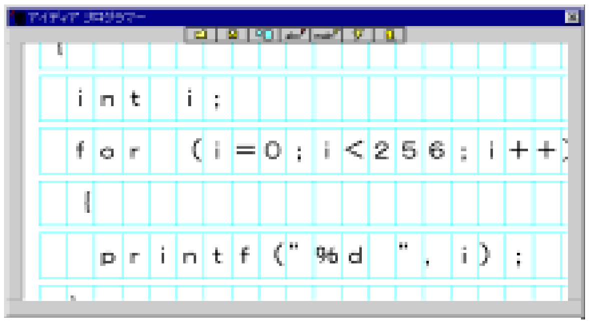
Since the electronic whiteboard has a limited size, and some programs may be too large to show within the display area, the scroll function is essential.

The standard scroll bar for a window is displayed on the bottom right-hand side of the window. This is convenient for operating in the traditional desktop environment, but on an interactive electronic whiteboard



```
{
 int  i;
 for  (i=0; i<256; i++
 {
  printf("%d ", i);
```

Figure 2. A handwritten program.



```
{
 int  i;
 for  (i=0; i<256; i++)
 {
  printf("%d ", i);
```

Figure 3. Result of program recognition.

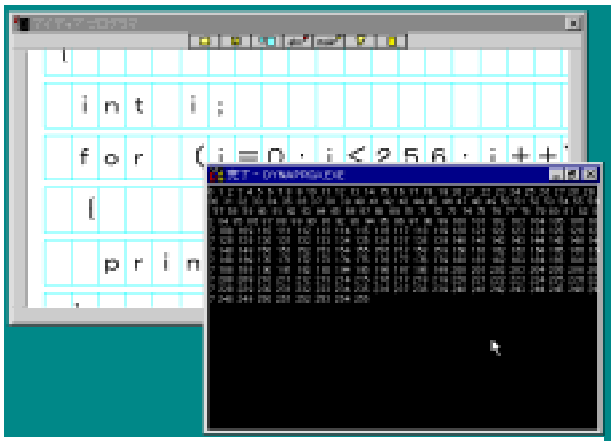


Figure 4. Result of program execution.

with a marker, it is hard to use. The teacher has to stretch his or her hands from side to side and hide the board by his or her body. This violates one of our basic design goals.

Therefore, we realize scrolling the screen without using the traditional scroll bar. Scroll area is located around the input area. By touching a marker on any place in this area and dragging it to arbitrary direction, the screen can be scrolled in that direction.

4.3. Handwriting annotations

A great advantage of using a pen is that one can write almost anything freely. It is very useful for the teacher to be able to write annotations on the program that he/she is explaining instead of merely showing the program. Therefore, our system provides an annotation capability on the program being explained by changing the mode of the pen from program writing to the annotation mode.

4.4. Editing

Input, insertion and deletion of program text are the most necessary editing functions. According to the design specifications, the user must be able to perform these operations with a single marker. We provide the input function by allowing the user to write a character pattern in any empty frame. Insertion and deletion can be performed by tapping a marker between lines and dragging right or left (insertion or deletion of character frames within a line), or down or up (insertion or deletion of lines). When a marker is dragged to the right, new frames appear along with the dragging and the user can write characters in them. When the marker is dragged to the left, the characters on the right move over the dragged characters which are deleted (Figure 5). Insertion and deletion of lines can be done similarly by shifting lines downward and making new lines or shifting lines upward. All of the above functions are executed smoothly.

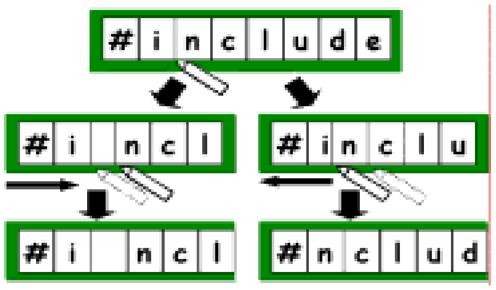


Figure 5. Insertion and deletion of text by sliding character frames.

Smoothness has been sought to realize move and copy functions as well. If you tap the marker between lines and wait for a certain period, then the color of the upper line is reversed. If you drag the marker up/down without detaching it, the color of more upper/lower lines is reversed. This shows that lines of text have been selected. Then, when you tap the marker at some place outside the reversed area, the selected lines are copied. On the other hand, if you tap the reversed area again, the selected lines are drawn into the marker smoothly. Then, when you tap the maker at some place, the saved lines are released there.

4.5. Handwritten program recognition

With a marker, writing is the easiest and simplest way to input program text. Without pattern recognition, however, handwriting is just pen-trace patterns and cannot be processed as program text. Therefore, handwritten character recognition is necessary.

Here, we follow the lazy recognition scheme which delays the display of recognition until needed. Lazy recognition also provides an easier structure to employ context processing [5].

When a teacher is writing a program and explaining it, machine recognition is not only unnecessary for the teacher and students, but it is even worse than when no recognition is employed. Recognition immediately after a pattern is written causes interruption because of the correction of incorrect recognition and the verification of recognition even if a pattern is correctly recognized.

When the teacher wants to show the execution of a handwritten program, however, program recognition is essential. Although the teacher might have to correct some misrecognitions, it is worth the trouble since the machine can show the execution of the program that the teacher has just written in front of the students.

The recognition rate of handwritten programs is truly much higher than usual text, since the constraints from the grammars of programming languages are very strong, so that the recognition rate is increased to almost the perfect level except user-defined identifiers only for that there is little constraints. The method of recognition is not the scope of this paper and described in [4, 6].

4.6. Defects and problems

There were some fatal defects and problems in the previous system:

(1) When a program requires input while running, it must be made from a keyboard or from a file.

(2) The output characters in the execution window were too small. This is due to the restrictions of the underlying DOS window.

(3) Annotations could not be made on the execution window. For explaining about the relation between a source program and its execution, however, it is better to be able to annotate over both the source codes and its execution results.

(4) We placed the menu of buttons at the center in the upper or lower screen frame so that the users can tap them from the both sides, but the users had to stretch their hands to tap distant buttons when the number of buttons becomes large.

5. Revised system

In this section, we will present the revised system focusing new features.

5.1. Program writing window

The layout of the program writing window in the revised system consists of a handwriting input area and a scroll area (Figure 6). The scroll area is located around the input area except for the upper part of the window.

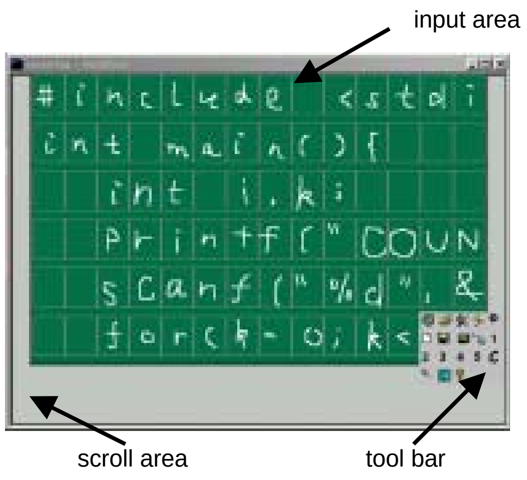
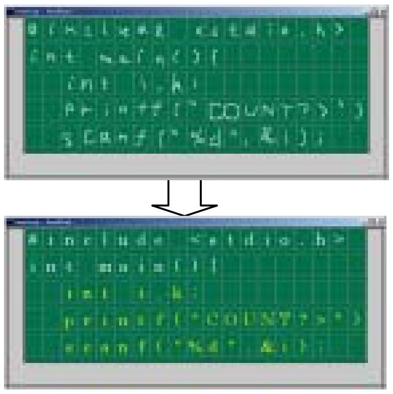


Figure 6. Revised screen layout.

When the user touches a marker anywhere in the scroll area and drags it to any direction, he or she can scroll the window to that direction.

When the user taps somewhere in the scroll area with the marker, the tool bar appears there. Since operations in the tool bar are not so frequently performed, the tool bar is displayed only when necessary. Moreover, only the buttons used often are displayed and others are hidden. When the functions associated with the hidden buttons are required, the number of buttons displayed can be changed by dragging the scroll area in the tool bar with the marker.

A user can write a program in the handwriting input area and make the system recognize it in the same way as the previous system as shown in Figure 7..



```
#include <stdio.h>
int main(){
    int i,k;
    printf("COUNT?>");
    scanf("%d",&i);
```

```
#include <stdio.h>
int main(){
    int i,k;
    printf("COUNT?>");
    scanf("%d",&i);
```

Figure 7. Handwritten program recognition.

5.2. Output of program execution

In order to have the control to the display of execution results, the output of the program being executed is not displayed directly, but a pipe with the system is created and the output is displayed within the execution window of the system explained later.

5.3. Data input

Now we can open an input area by tapping a menu button and write input data there. For this purpose, we employ the latest technology of writing-box-free handwriting recognition [3] so that the teacher can write input data without any writing grid.

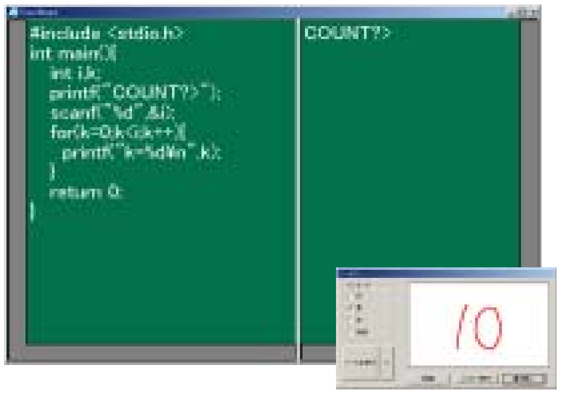
We have deliberately avoided the use of a keyboard for input in our system. A keyboard is neither easy to use for a standing teacher nor easy to observe for the students facing the teacher. When the teacher writes some input in the input area, its recognition result is displayed there. The teacher can confirm the result of recognition, correct it if necessary, and then make the program continue its execution by pushing the execution button (Figure. 8).

Input data thus recognized is sent to the running program through the pipe in the same way that the execution output is sent to the execution window.

5.4. Execution window

When the teacher is explaining a program, it is useful if he or she can show the source program and its execution results, and annotate them by writing as shown in Figure 9.

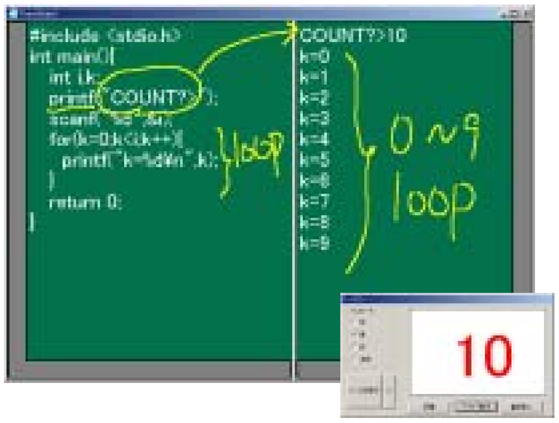
The execution window displays the source program and its execution result within split areas of the window and allows the user to annotate over both the areas as shown in Figure 9. On the other hand, the user can scroll each area separately. The user can choose either or both areas to be displayed, and can change their proportions of the window by dragging the splitter as shown in Figure 10.



```
#include <stdio.h>
int main(){
   int i,k;
   printf("COUNT?>");
   scanf("%d",&i);
   for(k=0;k<i;k++){
     printf("k=%d¥n",k);
   }
   return 0;
}
```

```
COUNT?>
```

Figure 8. Data Input and execution



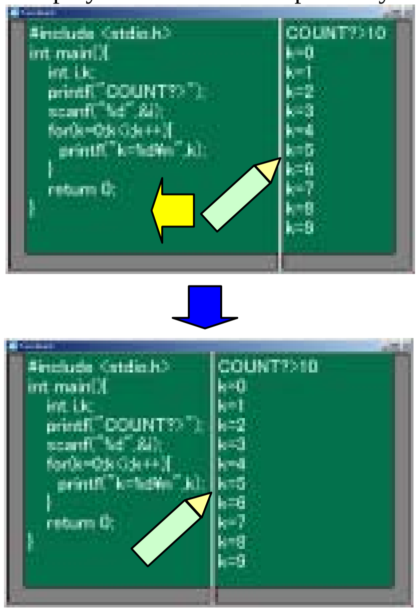
```
#include <stdio.h>
int main(){
   int i,k;
   printf("COUNT?>");
   scanf("%d",&i);
   for(k=0;k<i;k++){
     printf("k=%d¥n",k);
   }
   return 0;
}
```

```
COUNT?>10
k=0
k=1
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
```

Figure 9. Annotations made over both the areas.

Another alternative is to display the result of program execution in a window separate from the source program, but to allow the user to annotate over two or more windows requires restructuring of our system software for this system.

The tool bar containing buttons for operations on the execution window appears whenever the execution window is displayed because it occupies only a small area



```
#include <stdio.h>
int main(){
  int i,k;
  printf("COUNT?>");
  scanf("%d",&i);
  for(k=0;k<i;k++){
    printf("k=%d\n",k);
  }
  return 0;
}
```

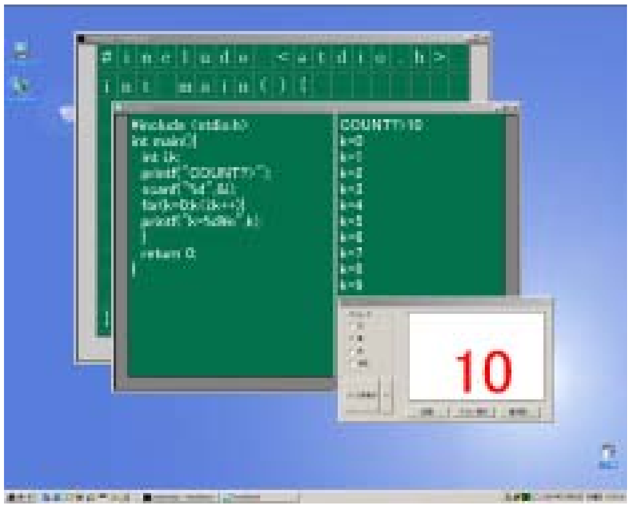
```
COUNT?>10
k=0
k=1
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
```

Figure 10. The splitter window displaying the source program and result.

and these operations are used often when the teacher is working on the execution window.

5.5. Implementation

We have implemented the revised programming education system on the MS-windows ME using Visual C++ 6.0. Its appearance is shown in Figure 11.



```
#include <stdio.h>
int main(){
```

```
#include <stdio.h>
int main(){
  int i,k;
  printf("COUNT?>");
  scanf("%d",&i);
  for(k=0;k<i;k++){
  printf("k=%d\n",k);
  }
  return 0;
}
```

```
COUNT?>10
k=0
k=1
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
```

Figure 11. Appearance of the revised system.

This paper described the design and implementation of our revised programming education system on an electronic whiteboard system. Using this system, a teacher can verify the correctness of a program that he/she has written immediately, show its execution, and explain how the output is changed when some part of the program is modified, without losing the familiarity and advantages of lectures using chalk and blackboard (Fig. 1).

We are now planning to use the system for teaching a programming course in our university curriculum to evaluate it in actual use.

7. References

[1] R. Carr and D. Shafer: *Power of PENPOINT*, Addison-Wesley. (1991).

[2] S. Elrod, R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang and B. Welch: "Liveboard: a large interactive display supporting group meetings, presentations and remote collaboration," *Proc. CHI'92*, pp.599-607 (1992).

[3] T. Fukushima and M. Nakagawa: "On-line writing-box-free recognition of handwritten Japanese text considering character size variations", *Proc. of 15th ICPR*, Vol.2, pp.359-363 (2000.9).

[4] J. Kanda, S. Sawada and M. Nakagawa: "Programming education system on an interactive electronic whiteboard," (in Japanese) *Proc. of 14th Symposium on Human Interface*, Tokyo, pp.601-606 (1998.9).

[5] M. Nakagawa, K. Machii, N. Kato, T. Souya: "Lazy recognition as a principle of pen Interfaces," *INTERCHI'93 Adjunct Proc.* pp.89-90 (1993.4).

[6] M. Nakagawa, K. Akiyama, L. V. Tu, A. Homma and T. Higashiyama: "Robust and highly customizable recognition of on-line handwritten Japanese characters," *Proc. 13th ICPR*, Vol. III, pp.269-273 (1996.8).

[7] M. Nakagawa, T. Oguni and T. Yoshino: "Human interface and applications on IdeaBoard," *Proc. INTERACT 97*, pp.501-508 (1997.7).

[8] M. Nakagawa: "Enhancing handwriting interfaces," *Proc. HCI International '97*, Vol. 2, pp.451-454 (1997.8).

[9] M. Nakagawa, K. Hotta, H. Bandou, T. Oguni, N. Kato and S. Sawada: "A revised human interface and educational applications on IdeaBoard," *CHI99 Video Proceedings and Video Program and also CHI99 Extended Abstracts*, pp.15-16 (1999.5).

[10] T. Oguni, T. Yoshino and M. Nakagawa: "Demonstration of the IdeaBoard interface and applications," *Proc. INTERACT 97*, pp.613-614 (1997.7).

[11] E.R. Pedersen, K. McCall, T.P. Moran and F.G. Halasz: "Tivoli: an electronic whiteboard for informal workgroup meetings," *Proc. INTERCHI'93*, pp.391-398 (1993.4).