プログラミング言語　13－4
（１９８７・９・１６）

視覚的プログラミングの試み

市川　忠男　、平川　正人
広島大学工学部

プログラミングは、専門的教育を受けたものにとっても、結構面倒で手間のかかる作業である。これは、本質的にはプログラミング環境の問題である。したがって、視覚的情報をプログラミングの過程に導入し、より親しみ易い環境を実現することによって改善することができる。
本文では、まずユーザにとって親しみ易いプログラミング環境の実現を目指したいくつかの試みを紹介し、その後で、筆者らが開発しているアイコニックなプログラミング環境ＨＩ－ＶＩＳＵＡＬの概要を述べる。
ＨＩ－ＶＩＳＵＡＬでは、データやプログラムなどのオブジェクトをアイコンで表わし、ディスプレイ上にこれらのアイコンを組合せ配置することによってプログラミングを行なう。プログラムの実行結果は直ちに表示され、プログラム開発はインタラクティブに進められる。なお、このインタラクション機能に加えて、大規模なプログラム開発に対応するためのトップダウンアプローチを支援する機能も用意されている。

Visual Programming - Toward Realization of User-Friendly Programming Environments

Tadao Ichikawa and Masahito Hirakawa

Information Systems, Faculty of Engineering, Hiroshima University
Shitami, Saijo-cho, Higashi-Hiroshima 724 Japan

Programming is generally considered to be a complicated and time-consuming task even for professionals. This comes mostly from the unfriendliness of the programming environment. One of the possible ways to overcome this problem is by the utilization of visual information in programming.

In this paper, we first observe some trials toward realization of user-friendly programming, and then give an outline of an iconic programming environment, HI-VISUAL, which the authors are now developing.

In HI-VISUAL, objects dealt with by the system, such as data and programs are represented in terms of icons. Programming is carried out simply by arranging icons on the display. The system displays intermediate results of execution in the course of program development so that the user can proceed program development interactively. In addition to this interaction facility for programming, program development based on the top-down approach is also provided for programming in the large. Details of the program modules can be left unspecified until they become clear in the lower level of module hierarchy.

1. Introduction

Programming is generally considered to be a complicated and time-consuming task even for professionals. This calls for an improvement of programming languages, and a number of programming language features valuable for the efficient production of reliable programs have already been identified through tremendous efforts directed toward to program development [1]. These features include data abstraction, modularization, concurrent programming, user-defined data type and type checking, and exception handling. Furthermore, recent studies on high level programming languages have contributed to the development of sophisticated languages based on new programming paradigms such as functional programming, logic programming, and object-oriented programming.

In addition to the improvement of program languages, the most important is the development of programming environments with many useful tools for programming such as syntax-directed editors and debuggers [2]. This has contributed to an increase in software productivity, a reduction of software costs, and also to the enhancement of program reliability.

The recent dramatic progress of computer applications thus attained, in turn, calls for user-friendly interfaces to enable people who are not familiar with computers to make programs. One of the possible ways for attaining this is the utilization of visual information in programming [3]-[5]. For example, a program itself and/or execution behavior of programs could well be visualized on the display.

In this paper, we first observe some trials toward realization of user-friendly programming, and then give the outline of an iconic programming environment, HI-VISUAL, which the authors are now developing [6]-[8].

HI-VISUAL which stands for HIroshima-VISUAL was first proposed as a language supporting visual interaction in programming [6]. Next, we presented design and implementational issues of HI-VISUAL with the objective of achieving interactive iconic programming [7]. Furthermore, we extended HI-VISUAL as a programming environment by providing several useful facilities for iconic programming [8].

In HI-VISUAL, objects which the system deals with such as data and program are represented in terms of icons. Programming is carried out simply by arranging icons on the two-dimensional display screen and specifying flow of data in the following way: The user selects an icon from the icon menu, and locates it at an appropriate place in the screen. When the icon becomes executable, the system activates it to execute the associated function immediately and returns the resultant data to the user. Programming proceeds by referring to the resultant data and connecting another icon to it.

In addition to the interaction facility for programming described above, program development based on the top-down approach is also supported in HI-VISUAL for programming in the large. Details of the program modules can be left unspecified until they become clear in the lower level of module hierarchy.

The interactions between the user and the system will be explained by showing several computer displayed pictures for ease of understanding. Following these examples are some remarks toward the development of generalized iconic programming system to be applied to a variety of computer applications.

2. User-Friendliness

Of the many trials being carried out at present for the development of user-friendly programming, those which utilize visual information look especially promising [3]-[5]. The user makes a program through visual interaction with the system. This kind of interaction scheme can be broadly termed a 'visual language.'

In visual languages, information to be visualized is either (i) an object such as data, file, and program, (ii) an algorithm of the program, or (iii) a data structure. Figure 1 shows the classification of visual languages. The categories (i), (ii), and (iii) described above are used again in the figure for identification.

The most remarkable feature of the systems categorized in (i) is the visualization of objects by means of icons [9]. In these systems, the user specifies the job by 'see and pointing' of icons on the display screen.

In the systems categorized in (ii), the user makes a program using flow charts, Nassi-Shneiderman (NS) charts, Problem Analysis Diagrams (PAD), data flow graphs, state transition diagrams, etc. The graphic representation of programs is translated into an internal representation recognizable by a conventional computing system. SDL/PAD [10] and the state transition diagram language [11] are examples of languages which support program development. Another example is OPAL [12] which was designed to specify procedural knowledge for an expert system.

The systems categorized in (iii) visualize data structures by means of forms or graphics. QBE [13] and form languages [14], [15] are examples. The user specifies the job by filling in spaces on a form with conditions. Another example is a browser for the manipulation of knowledge-base/database [16]. The knowledge-base/database is manipulated graphically along the link indicating a particular relationship between data items. Incense [17] and VIPS [18] are also examples of this category, which have been designed to make debugging easier with the help of graphical representation of data structures on the display.

There are also systems which work with combinations of categories (i), (ii) and (iii).

Examples of systems combining categories (i) and (ii) are Pict [19] and Tinkertoy [20] which are applied in a general programming environment, construction game kits (pinball games, for example), IBS [21] for manipulating relational database by means of an icon-based command language, and PegaSys [22] for supporting program design.

ISIS-V [23] is an example of a system which combines categories (i) and (iii) in a database environment.

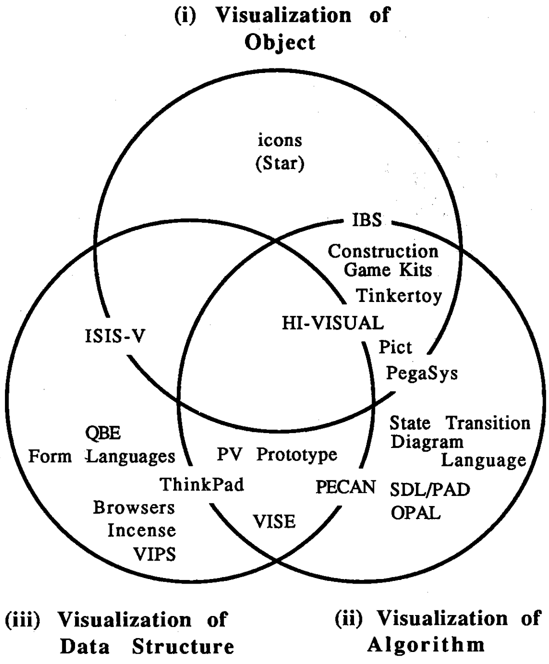

Fig. 1 Classification of visual languages

ThinkPad [24] combines categories (ii) and (iii), and allows the user to specify a program by demonstration. Other examples are PECAN [25], VISE [26], and PV Prototype [27], which were designed to ease debugging and program understanding, in which visual information is applied so that the user can realize the appropriateness of program behaviors visually. This is referred to as 'program visualization [4].

In contrast to this, the way of utilizing visual information to specify a program in two-dimensional space is referred to as 'visual programming [4].' HI-VISUAL [6]-[8] is a visual programming environment and combines categories (i), (ii), and (iii). In HI-VISUAL, object, algorithm, and data structure are visualized by means of the icon, data flow graph, and spatial placement of icons, respectively.

3. Icons

3.1 Icon Definition

Most people would probably interpret the technical term 'icon' to mean simply a small picture or image on a display which is used to assist communication between the user and the system. The people who develop iconic systems, however, see it as a concept including both an object consisting of an icon image displayed on the screen and the functional description associated with it such as a program code and a data value.

Korfhage and Korfhage [28] discussed characteristics of icons and some related concepts exploring the possibility of developing iconic interface systems. They classified icons into two types as follows: One is an 'object icon' representing an entity, and the other an 'action icon' representing a specific action. In addition, they also have presented concepts for construction of structured agglomerations of icons.

Chang [29] proposed a generalized icon which can take either object icon or action icon (called 'process icon' in his paper). The distinction between an object icon and an action icon depends on the application context and its interpretation.

Furthermore, an extended interpretation of the icon is presented in [30], where an icon can be associated with a particular feature shared by objects of different shape. For example, in an engineering drawing application, a group of parallel lines denoting a section of an object is considered to be an icon. Here the icon form varies depending on the shape of the objects.

3.2 Icon Semantics

Concerning icon semantics, we will restrict our discussion specifically to those of HI-VISUAL. The objects the system deals with are recognized as icons. An icon has both an internal part and an external part [8]. The internal part gives the meaning of an object, and has three attibutes: *substance*, *concept*, and *type*. The external part specifies the visual representation of an object, and has three attributes: *image*, *label*, and *shape*.

Substance represents the functional description or value of an object. *Concept* gives the conceptual name of an icon, and is provided for semantic-based management of icons. The system allows the user to make a program easier by showing icons which can be used at each step of programming. *Type* represents the type of an icon. There are seven types, as shown in Fig. 2.

(i) DATA represents data such as characters, numbers, and image.

(ii) DATA CLASS represents the class of data. It is similar to the class in the object-oriented concept, and manages not only a data type but also functions applicable to data items which belong to the data type. Data classes are managed hierarchically according to their semantics, and then functions are inherited along the data class hierarchy.

(iii) PANEL represents a display space for the management of a set of icons. It is similar to the directory of conventional file systems. A window and a menu are regarded as the panel icon.

(iv) PROGRAM represents a program which is constructed by the user as a combination of existing icons. Unlike PANEL, icons which construct the program must be connected to each other.

(v) PRIMITIVE represents a basic program which is prepared by the system designer as a built-in function. Both program icons and primitive icons will hereafter be referred to as function icons.

(vi) CONTROL represents the control of a flow of data, such as loop and switch.

(vii) COMMAND represents a command for system operations, such as run, undo, and exit.

Concerning the descriptions of the external part, *image* represents the icon image which symbolizes an object. *Label* represents the name of an icon. *Shape* represents the shape of the frame of an icon. Seven different shapes are defined depending on the icon type, which are also illustrated in Fig. 2.

4. HI-VISUAL

In HI-VISUAL, object, algorithm, and data structure are visualized by means of an icon, a data flow graph, and a spatial placement of icons, respectively.

4.1 Icon Programs

A HI-VISUAL icon program takes the form of a sequence of combinations of a 'function icon' and a 'data (or data class) icon.' Each data icon represents the result of execution by a function icon. Function icons are allowed to have only one input and one output. An input/output data (or data class) may have a record structure as shown in Fig. 3.

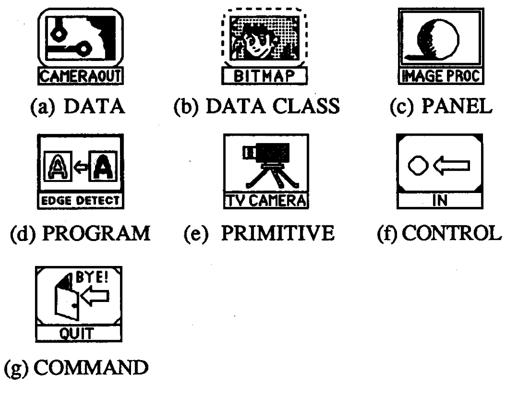

Fig. 2 Icon type

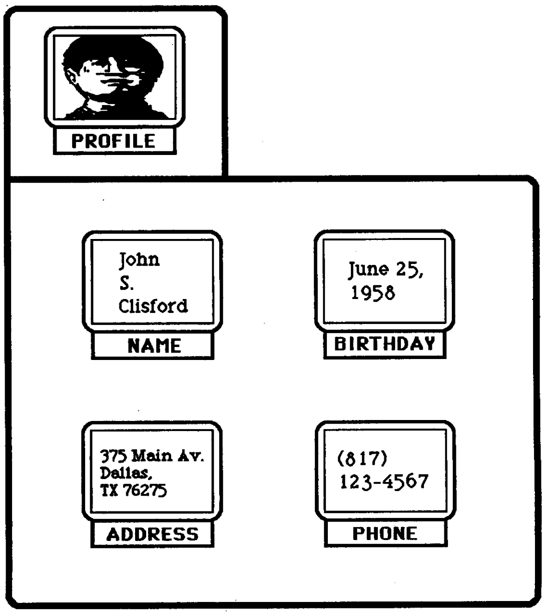

Fig. 3 An example of record-type data

Figure 4 shows an example of the icon program, which is an image processing routine for the detection of cracks in an input image. The function specified by this program is as follows: First, the image which is taken from the TV camera is binarized by the application of BINARIZE icon. The binarized image is applied to both CRACK DETECT and EDGE DETECT icons. The resultant images are combined into a single icon, which is applied to SYNTHESIZE icon which performs logical OR. SYNTH. OUT is the final result obtained on completion of the program execution.

HI-VISUAL also provides two types of control icons: a 'loop icon' for specifying the iteration of procedures and a 'switch icon' for specifying the change of the flow of data. Loop icons are classified again into a 'counter loop' and a 'conditional loop.'

In the counter loop, the number of iterations is specified by using a data icon. Figure 5 shows an example of the counter loop in which a value of the 5th power of 2 is calculated. The process specified in the square is repeated until the number of iterations reaches the iteration condition. Two input data (initially, both set to 2) are multiplied with each other. The intermediate result and the data which is passed from an input are brought back to the inputs. When this process was repeated 4 times, the program outputs the final result (value of 32).

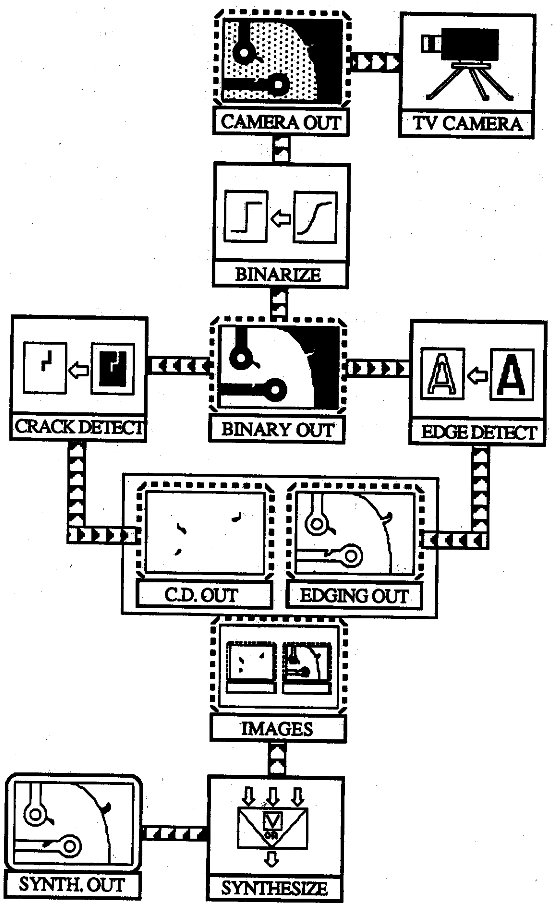

Fig. 4 An example of icon program

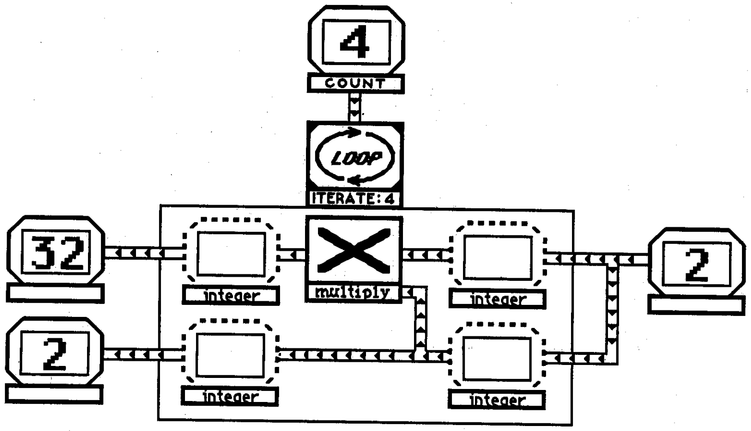

Fig. 5 An example of counter loop icon

Figure 6 shows an example program for neutralizing a sample liquid, in which a conditional loop icon and a switch icon are used. The condition parts of these control icons are specified by other icons as shown in the figure. If the comparison condition in the switch icon is true (the sample is alkaline), input data to the switch icon is transferred to the icons connected to the v-side of a switch, and acid is added to the sample. Otherwise (the sample is acid), input data is transferred to the icons connected to the x-side of a switch, and alikaline is added to the sample. The conditional loop activates the process repeatedly untill the sample becomes neutral.

In iconic programming systems, since an icon program is specified by the arrangement of existing icons, the applicability of the system basically depends on the set of primitive icons provided. If the primitive icons for image processing are available, the programs for other image processing applications can be developed by the system. If the primitive icons for office information processing are available, the programs for execution of higher level office

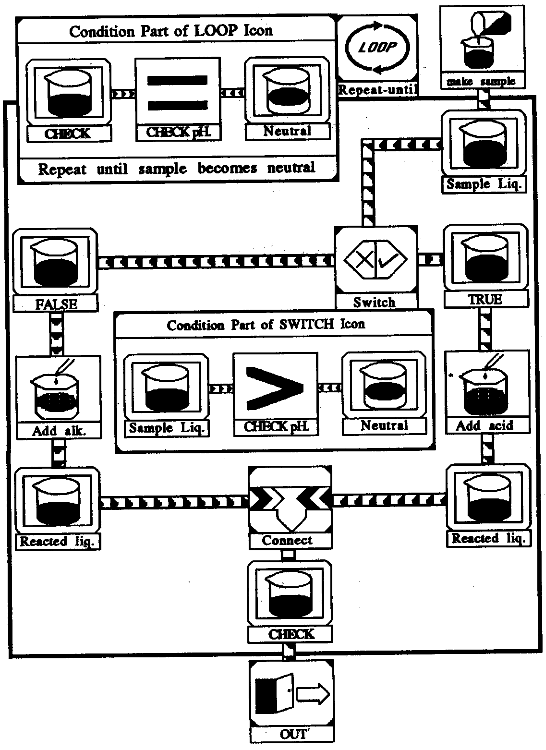

Fig. 6 An example of condition loop icon and switch icon

procedures can by developed by the system. In HI-VISUAL, a tool for defining primitive icons is provided [7] enabling the system be applied to various application environments.

4.2 Iconic Programming

HI-VISUAL provides an interactive, icon-based programming facility. In the following, we explain how the programming is carried out in HI-VISUAL.

First, the user selects an icon from the menu of icons by using a pointing device such as a mouse, and locates it at a proper place on the screen. If the icon is a function icon, the output from the icon is presented to the user. At this time, when the icon is executable, the system activates it immediately to execute the associated function and displays the miniaturized representation of the resultant data. On the other hand, if the function icon is not executable, the system displays a data class of the resultant data. Here, the term 'executable' means the occurrence of either 1) no input data of the icon is needed or 2) all input data has already been provided.

Next the user selects and places another icon on the screen, and specifies connections between icons to form the necessary flow of data. The system executes the icon and displays the resultant data on the screen in way described above.

The procedures described above are repeated until the program development is completed. The user can make a program by confirming the program behavior interactively at every step of program development. If the result does not meet the user's requirements, the user can replace the icons previously specified with others.

4.3 Bottom-Up and Top-Down Development

The system supports both the bottom-up and top-down approaches for program development.

Suppose the user has created a program by arranging existing icons as we described in the previous section. In the bottom-up approach, the program is defined as a new (program) icon at a higher level of program abstraction, and it is used to create a new program. This approach is effective especially when the system is applied to an environment where tasks are not well defined in advance of the program development. However, since interpretation and execution of icons are carried out repeatedly during the process of program development, the execution speed decreases. These problems seem to be serious when the size of a program is fairly large.

Capability of program development based on the top-down approach is provided as a countermeasure to it. Details of the program modules can be left unspecified until they become clear in the lower level of module hierarchy. Figure 7 shows an example of top-down program development.

The user first makes an upper level program by defining input/output parameters (data classes) of program icons in a lower level and specifying connections between them. Small windows appear besides the program icons for the specification of program modules. Next, when the user clicks a program icon, a new window appears in which details of the program in a lower level are specified. Here input and output data classes are presented to the user so that the consistency is kept on data classes of upper and lower level programs.

4.4 Navigation Facilities

To ease program development, a navigation facility for iconic programming is provided in the system. The system displays a list of all candidate icons which can be used at each step of program development. This navigation facility also helps to prevent the user from making illegal specification of a program.

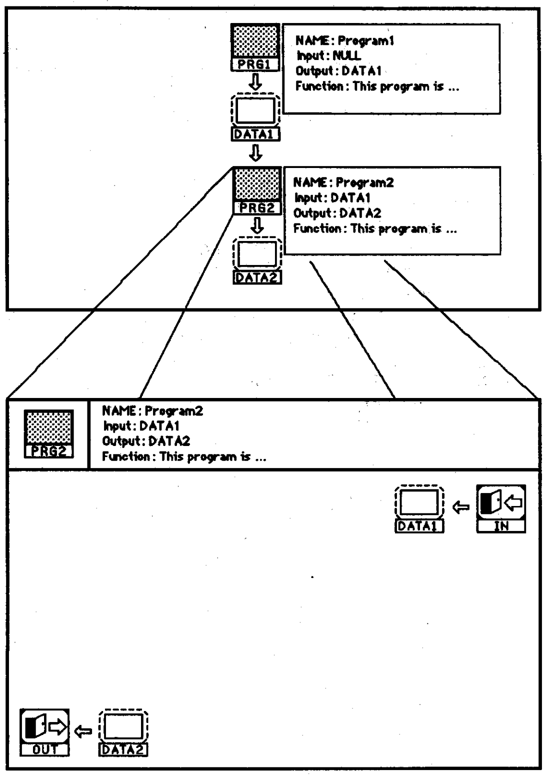

Fig. 7 An example of top-down programming

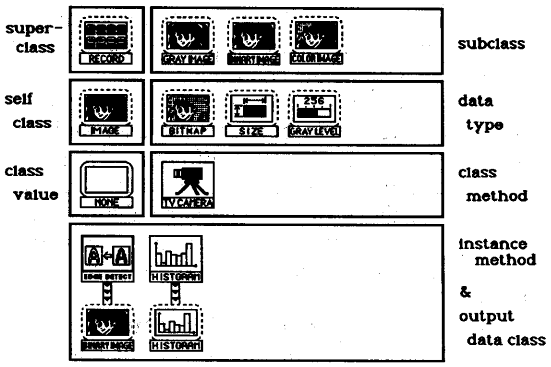

Fig. 8 Icon-based browser

Figure 8 shows a display of an icon-based browser [8] which has been provided in the system for programming navigation. The user selects a particular data icon or data class icon, and invokes the browser. The selected icon itself is displayed in the self class field. The instance method & output data class field shows function icons which can be connected to the selected icon. Function icons requiring no input data are displayed in the class method field. Superclass and subclasses of the data class are displayed in the superclass field and subclass field respectively. Class value of the data class is displayed in the class value field.

The user selects an icon displayed in the browser, and continues the programming. If the expected icon is not found, the user selects a superclass icon in the superclass field and then browses through the class hierarchy until he gets the expected function icon.

In addition to this navigation facility for programming, the system also provides another navigation facility for system operation which works in cooperation with a knowledge-base system. It helps the user know (i) the system's response expected against the user's action, such as pressing the mouse button and moving the cursor, and (ii) the system commands which can be used at a current cursor position. The user is actually informed of both (i) and (ii) by the system.

5. Concluding Remarks

In this paper, we first observed trials for attaining a user-friendly programming environment through the utilization of visual information, and then gave the outline of an iconic programming environment, HI-VISUAL, now being developed by the authors.

In HI-VISUAL, icons represent objects dealt with by the system such as data and programs.

Programming is carried out simply by arranging and connecting icons on the display. Data is passed between icons along the connection.

Icons provide an effective means of improving interaction between the user and the computer from the viewpoints of both universality and efficiency. Icons are easily understood by people regardless of the language they speak. They are sometimes ambiguous, however, depending on the social/technical background of the user.

One successful approach for overcoming this disadvantage is to limit the domain of applying iconic systems so that the users can share a common background. In HI-VISUAL, a tool for defining primitive icons is provided. The system designer can define primitive icons for a specific application. This tool enables the system to be applied to various environments such as office information processing, image processing, simulation, and CAI. Furthermore, a knowledge-base system should be investigated as the next step of our research, so that flexible interpretation of icons can be made feasible depending on the environment.

For future works toward realization of generalized iconic programming environment, we should aim at development of a useful environment not only for casual users but also for professional programmers. The environment should provide sophisticated tools for programming in the large, facilities for visualizing the dynamic behavior of a program through animation, and multiple views of a program for ease of program development/ debugging. Furthermore, we need to investigate a compilation scheme of icon programs to make the system work in an actual environment by speeding up the execution of programs.

In conclusion, HI-VISUAL is expected to serve programming in various fields of application supporting simple and efficient interactions between the user and the computer.

REFERENCES

[1] J. W. Hunt, "Programming Languages," IEEE Computer, Vol. 15, No. 4, pp. 70-88, Apr. 1982.

[2] E. Fedchak, "An Introduction to Software Enginnering Environments," Proc., IEEE COMPSAC'86, pp. 456-463, 1986.

[3] S. K. Chang, T. Ichikawa, and P. A. Ligomenides (eds.), Visual Languages, Plenum Press, New York, 1986.

[4] B. A. Myers, "Visual Programming, Programming by Example, and Program Visualization: A Taxonomy," Proc., CHI'86, pp. 59-66, 1986.

[5] S. L. Tanimoto and E. P. Glinert, "Designing Iconic Programming Systems: Representation and Learnability," Proc., IEEE Workshop on Visual Languages, pp. 54-60, 1986.

[6] N. Monden, Y. Yoshino, M. Hirakawa, M. Tanaka, and T. Ichikawa, "HI-VISUAL: A Language Supporting Visual Interaction in Programming," Proc., IEEE Workshop on Visual Languages, pp. 199-205, 1984.

[7] I. Yoshimoto, N. Monden, M. Hirakawa, M. Tanaka, and T. Ichikawa, "Interactive Iconic Programming Facility in HI-VISUAL," Proc., IEEE Workshop on Visual Languages, pp. 34-41, 1986.

[8] M. Hirakawa, S. Iwata, I. Yoshimoto, M. Tanaka, and T. Ichikawa, "An Environment for HI-VISUAL Iconic Programming, " Proc., IEEE Workshop on Visual Languages, 1987 (to appear).

[9] K. N. Lodding, "Iconic Interfacing," IEEE Computer Graphics and Applications, Vol. 3, No. 2, pp. 11-20, Mar./Apr. 1983.

[10] H. Maezawa, M. Kobayashi, K. Saito, and Y. Futamura, "Interactive System for Structured Program Production, Proc., Conf. on Software Engineering, pp. 162-171, 1984.

[11] R. J. K. Jacob, "A State Transition Diagram Language for Visual Programming," IEEE Computer, Vol. 18, No. 8, pp. 51-59, Aug. 1985.

[12] M. A. Musen, L. M. Fagan, and E. H. Shortliffe, "Graphical Specification of Procedural Knowledge for An Expert System," Proc. IEEE Workshop on Visual Languages, pp. 167-178, 1986.

[13] M. M. Zloof, "QBE/OBE: A Language for Office and Business Automation," IEEE Computer, Vol. 14, No. 5, pp.13-22, May 1981.

[14] N. C. Shu, "FORMAL: A Forms-Oriented Visual Directed Application Development System," IEEE Computer, Vol. 18, No. 8, pp. 38-49, Aug. 1985.

[15] S. B. Yao, A. R. Hevner, Z. Shi, and D. Luo, "FORMANAGER: An Office Forms Management System," ACM Trans. on Office Information Systems, Vol. 2, No. 3, pp. 235-262, July 1984.

[16] M. Caplinger, "Graphical Database Browsing," Proc., ACM SIGOIS Conf. on Office Information Systems, pp. 113-121, 1986.

[17] B. A. Myers, "Incense: A System for Displaying Data Structures," Proc., ACM SIGGRAPH'83 Conf., pp. 115-125, 1983.

[18] S. Isoda, T. Shimomura, and Y. Ono, "VIPS: A Visual Debugger," IEEE Software, Vol. 4, No. 3, pp. 8-19, May 1987.

[19] E. P. Glinert and S. L. Tanimoto, "Pict: An Interactive Graphical Programming Environment," IEEE Computer, Vol. 17, No. 11, pp. 7-25, Nov. 1984.

[20] M. Edel, "The Tinkertoy Graphical Programming Environment," Proc., IEEE COMPSAC'86, pp. 466-471, 1986.

[21] C. Frasson and M. Er-radi, "Principles of An Icons-Based Command Language," Proc., ACM SIGMOD Conf., pp. 144-152, 1986.

[22] M. Moriconi and D. F. Hare, "The PegaSys System: Pictures as Formal Documentation of Large Programs," ACM Trans. on Programming Languages and Systems, Vol. 8, No. 4, pp. 524-546, Oct. 1986.

[23] J. W. Davison and S. B. Zdonik, "A Visual Interface for A Database with Version Management," ACM Trans. on Office Information Systems, Vol. 4, No. 3, pp. 226-256, July 1986.

[24] R. V. Rubin, E. J. Golin, and S. P. Reiss, "ThinkPad: A Graphical System for Programming by Demonstration," IEEE Software, Vol. 2, No. 2, pp. 73-79, Mar. 1985.

[25] S. P. Reiss, "PECAN: Program Development Systems that Support Multiple Views," IEEE Trans. on

Software Engineering, Vol. SE-11, No. 3, pp. 276-285, Mar. 1985.
[26] A. K. Arora, D. K. Chan, J. C. Ferrans, and R. Gordon, "An Overview of The VISE Visual Software Development Environment," Proc., IEEE COMPSAC'85, pp. 464-471, 1985.
[27] G. P. Brown, R. T. Carling, C. F. Herot, D. A. Kramlich, and P. Souza, "Program Visualization: Graphical Support for Software Development," IEEE Computer, Vol. 18, No. 8, pp. 27-35, Aug. 1985.
[28] R. R. Korfhage and M. A. Korfhage, "Criteria for Iconic Languages," in *Visual Languages*, S. K. Chang, T. Ichikawa, and P. A. Ligomeniddes (eds.), pp. 207-231, Plenum Press, New York, 1986.
[29] S. K. Chang, "Visual Languages: A Tutorial and Survey," IEEE Software, Vol. 4, No. 1, pp. 29-39, Jan. 1987.
[30] M. Beretta, P. Mussio, and M. Protti, "Icons: Interpretation and Use," Proc., IEEE Workshop on Visual Languages, pp. 149-158, 1986.