Visualized and Modeless Programming Environment for Form Manipulation Language

Jun'ichi Miyao, Shin'ichi Wakabayashi, Noriyoshi Yoshida, and Yasushi Ohtahara
Faculty of Engineering, Hiroshima University
Higashi-Hiroshima 724, Japan

Abstract

A visualized and modeless programming environment VPF is presented in this paper. The VPF supports the development of programs written in a programming language, called a form manipulation language. The language is a so-called iconic language, and provides not only programming facilities but also form-oriented operations. The VPF provides the following features: (1) hierarchical views that support views for various abstract levels of program specification and execution. (2) visualization of program itself and its run-time behavior with graphical representation, and (3) modeless operations that enable us to develop a program under only one execution mode. These features give the programmers good comprehension of program development, and enhance productivity and reliability of the application programs.

1 Introduction

In office processing, forms play an important role. They are suitable interface for information processing, because most information in the office is represented as forms. Thus, office systems should provide a form interface which enables us to interact with the system through forms. Recently, many researches with respect to forms have been proposed [Hirakawa87, Hirakawa88], and also many form systems that employ forms as user interfaces are developed such as OBE [Zloof82], FORMAL [Shu88], Browsing System [Larson86], CRAFT [Hirakawa88], etc. However, there are few systems which provide sufficient programming capability to describe an application program or a series of from processing.

Therefore, we have proposed a form manipulation language which is a special purpose iconic language, and provides not only programming facilities, but also form-oriented operations [Miyao87]. The language is special purpose, but there exist general purpose visual languages such as HI-VISUAL [Hirakawa87], Pict [Glinert84], etc. However, our language obtains high usability instead of losing general purpose facilities.

2 Form Manipulation Language

2.1 Form Operations

In this subsection, we present forms and operations on them. At first, we explain forms intuitively by using examples. Figure 1 (a) and (b) show examples of an examination result and announcement for supplementary report, respectively. Each entry in the forms is filled with proper information which is entered by user or retrieved from database. In form systems, there are two kinds of user processing, form definition and form manipulation.

In the in the paper, we propose a visualized programming environment called VPF (Visualized Programming Environment for Form Manipulation Language) for the form manipulation language. Main features of the environment are summarized as follows:

- Hierarchical view support: In the program development by form manipulation language, the system supports four abstract levels of programming from screen of application programs to database instances.
- Visualization of program behavior: For each level, a program and its run-time behavior are visualized.
- Modeless operations: Users can develop programs under only one mode. All operations are performed under the execution mode.

Above features give the programmers good comprehension of program development, and enhance productivity and reliability of the application program.
2.2 Form Manipulation Language

So far, various languages for form manipulation are proposed in OBE [Zloof82], FORMAL [Shu88], Browsing System [Larson86], CRAFT [Hirakawa88], etc. However, most of them are developed for the ad hoc queries. The form manipulation language should provide capability of describing fixed queries, because most processing for forms in the real world are executed repeatedly.

Therefore, we have proposed a form manipulation language in [Miyao87]. We illustrate outline of the language and summarize its features.

A form manipulation language $F_M$ is defined by $F_M = (F_D, P_G, I_F)$ where $F_D$ is a form definition, $P_G$ is a program graph and $I_F$ is a set of icon forms. In the following, we explain these three elements.

The form definition is defined by a form type $F_D = (F_L, F_V, F_C)$ where $F_L$, $F_V$ and $F_C$ are a logical scheme, a visual template and a connection, respectively. The connection specifies relationships between entries in a form and attributes in relations. In the form definition, $F_L$ and $F_V$ are specified by drawing a form on a display screen as it is, and $F_C$ is specified by using domain variables like as OBE [Zloof82] (see Figure 2). In the form manipulation language, forms which are used in a program, must be defined in advance.

An application program of the form manipulation language is specified by a connected digraph $P_G = (V_P, E_P)$, called a program graph, and a set of icon forms $I_F$. The program graph $P_G = (V_P, E_P)$ must satisfy the following conditions (a)-(e).

(a) $V_P$ is a set of nodes, each of which has an identifier. Each node belongs to one of the following eight types: Form.Fill.In, Query, Display Form, Get, Sort, Message, Start and End.

(b) $E_P$ is a set of directed edges. Each edge belongs to one of two types: a normal edge represented by black line and an error edge represented by white line.

(c) $P_G$ contains exactly one Start node. Start node has no incoming edge.

(d) $P_G$ contains at least one End node. Each End node may have several incoming edges, but no outgoing edge.

(e) Each node on $P_G$ except for End node has exactly one outgoing normal edge, and at most one outgoing error edge. And, each node except for Start node has at least one incoming edge.

For each node in $P_G$, an icon form must be specified to define detailed operations in the node. $I_F$ is a set of these icon forms.

In the program graph, nodes represent operations, and directed edges represent control flows. Then, the program is executed from Start node to End node along normal directed edges (black lines). However, when an error is occurred at a node, the program is executed along the error outgoing edge (white line) of the node. If the node has no error outgoing edge, then the program is terminated (It is assumed that an error outgoing edge to End node exists.).
Figure 3 shows a program graph for the application program which makes an examination result and announcement for a supplementary report. Now, the semantics for the icons in this program graph will be explained. Start and End nodes represent start and end of an application program, respectively. Form.Fill.In node supports data entry by user and arithmetic and logical operation for the instance value described in the form. Query node deals with database operations. Display.Form and Message nodes display the form filled with data and message to users, respectively. And Figure 4 shows an example of the icon.form for the Form.Fill.In node.

The features of the form manipulation language are summarized as follows:
- **Visual and special purpose language**: Iconic language provides visual and intuitive programming capabilities. It is easy to understand and to use for end users compared with general purpose languages.

- **Form-oriented language**: The form model, which underlies the form manipulation language, allows to define forms with arbitrary two dimensional format.

- **Sufficient database operations**: This language adopts a QBE-like query language and thus provides sufficient database operations.

### 3 VISUALIZED AND MODELESS PROGRAMMING ENVIRONMENT VPF

#### 3.1 Features of VPF

Recently, many programming environments have proposed such as Smalltalk, Cedar, PECAN, etc. However, there still remain the following problems.

- There exist few environments to show the program behavior with real data visually and dynamically.
- There still exists a drawback that the programmers have bothered to change modes between editing and testing.

We propose a visualized programming environment called VPF to resolve the above problems by introducing hierarchical view of the program execution, visualization of program behavior, and modeless operations.

In the following, we present the features of VPF.

1. **Hierarchical view of the program execution**

   - Reference to multi-level: VPF provides the programmers with multiple consistent windows corresponding to four abstract levels in the program execution, i.e., user level, program graph level, icon.form level, data level. These windows can be displayed on the screen simultaneously, and each of them can also be represented in an iconic form. Since all the processes corresponding to each level are executing simultaneously in each window, programmers can develop a program, referring to any window at once.

   - Level-directed editors: VPF offers a custom editor for each level. All the editors allow the programmers to build the program by using a mouse and pop-up menu. These tools are used not only to select a command but also to manipulate object at each level, for example, a program graph is built by pointing icons, and selecting commands from the menu.

2. **Visualization**

   - Detection of run-time errors: Since programmers can build a program with visualized representation of each level, they can easily and intuitively detect where the run-time error occurred.

   - Detection of quitted point: When the program quit on account of syntax errors or wrong input data, etc., programmers can immediately detect where the program is quitted.
(3) Modeless operations

- Editing under execution: Programmers can edit a program while the program is being executed. VPF has only one mode, i.e., execution. In order to realize editing under execution, we introduce the concept "speed". Speed "0" implies that the program execution is paused.

- Execution from an arbitrary icon: Programmers can start the program from any icon in the program graph. If a pointed icon does not have enough input data, the program requests user's input. If the control of execution reaches to the icon which has incomplete parameters, the program also pauses at this icon.

- Automatic reexecution: If the programmers modify the parameter of an icon that was already executed, the program is executed again automatically from the icon which was modified.

- Fast turn-around time: VPF realizes short turn-around time of the cycle: modify to execute. Programmers can immediately look at the execution result effected by the modification of the program.

(4) Other features

- Visual user interface: VPF provides the programmers with multi-window, pop-up menu and a pointing device mouse.

- Speed control facility: Programmers can make the execution speed up or down at any time during the program development. This is done by pointing a speed indicator displayed on the screen. It is useful for programmers to execute a program icon by icon.

3.2 Configuration of VPF

In this subsection, software and hardware configurations of VPF are explained briefly.

(a) Software Configuration

Figure 5 shows the software configuration of the system VPF. The system consists of eight components and is written by C language on UNIX operating system SunOS3.5 except for Screen Manager. The Screen Manager is implemented on the Sun window system NeWS based on PostScript language.

(b) Hardware Configuration

VPF is implemented on a workstation Sun-3/60c (1180 x800 color display) with 4 MB main storage and 141 MB hard disk unit.

4 PROGRAM DEVELOPMENT IN VPF

In this section, we explain the program development in VPF by using an example of form processing.

[Example 2] Now, we show how to write an application program for the form processing to construct an examination result and announcement for supplementary report.
screen (see Figure 8). We select an icon displayed in the ICON field with the mouse, and put it on the PROGRAM GRAPH field. We construct the program graph, repeating this process. Figure 8 shows a part of the program graph that we are constructing.

**Display of I-level window**: In order to specify the parameters of an icon form for each icon, we point at one icon in the program graph with the mouse. Then, the I-level window appears on the screen, and the icon form for the pointed icon is displayed in this window. Then, we specify the parameters by using the mouse and the pop-up menu. We can specify the icon form at any time even if the program graph is not complete. Figure 9 is an example of the icon form for I8.
Display of U-level window: We open U-level icon and set a proper speed by pointing the indicator to execute the program. Looking at the U-level window, we can confirm whether the program works correctly or not. Figure 10 shows an example of U-level window while the program is executed. Now, we look at the result of the icon I14. The form displayed in the U-level window is the one that filled with the students who got a low grade D, i.e., less than 35 marks. However, we find the students whose marks are more than 35. Thus, the program contains some bugs.

Display of D-level window: We suppose that icon I12 causes this run-time error. In order to detect the cause of this error, we point the icon I12 to execute again from I12, and open D-level icon. Figure 11 shows an example of D-level window while the icon I12 is executed. This icon is the Query that retrieves name, major and marks of the students who take "compilers" as the course, and got the grade D. We look at the query in the QUERY field, and we can find that the value of attribute grade in table EXAM is an example element "-D". Thus, "-D" must be substituted for "D".

Correction of the error: In order to correct this error, we open I-level icon again, and correct "_D" in the table EXAM to "D".

Execution again: After we modify the icon form, we execute the program again. Now, we get a correct result of the program and complete the program.

Figure 11 Display of D-level window.

5 CONCLUSION

This paper has discussed the visualized programming environment for form manipulation language and its prototype system VPF. The main features of proposed programming environment are modeless operations, hierarchical view of program execution, and visualization of operations. Then, VPF enhances productivity and reliability of an application program.

Finally, we point out future research works.

(1) The forms in the real world have an arbitrary hierarchical structures. To build the prototype system VPF, however, we restrict the form model, i.e. we don't support all arbitrary hierarchical structures of forms. The formal form model presented in [Hirakawa88] allows a dynamic form of which logical structure is changed by its instance data. It is necessary to develop the facility which can deal with the dynamic forms.

(2) Programming environment VPF is a fixed environment, i.e. all programmers must use the same environment. If the programmers can customize the environment itself, they can construct their own environment. It is easy and friendly for individual programmers to use the customized environment.

REFERENCES


