ICOT Technical Memorandum: TM-0125

TM-0125

The Concepts and Facilities of SIMPOS Window System

by
J. Tsuji (Mitsubishi Electric Corp.)
and S. Uchida

July, 1985

©1985, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

The Concepts and Facilities of SIMPOS Window System

Junichiro Tsuji
(Mitsubishi Electric Corporation)

Shunichi Uchida
(Institute for New Generation Computer Technology)

Abstract

This report describes the window system of SIMPOS (Sequential Inference Machine Programming and Operating System). The window system, one of input/output medium systems, will provide the interface between PSI machine (Personal Sequential Inference machine) and the user through the bit-mapped display, keyboard and mouse.

Table of Contents

1. Introduction

SIMPOS [Hattori 84, Takagi 84] is a programming and operating system for PSI [Taki 84, Yokota 84]. It provides researchers with software development tools for logic programming. The operating system part of SIMPOS has three layers: the kernel, the supervisor, and the input/output medium systems.

The window system is one of the input/output medium systems. It manages the bitmapped display, the keyboard and the mouse, and it provides to the user the multi-window environment which is a high level man-machine interface. This report describes the concepts and facilities of the SIMPOS window system. Chapter 2 describes the various concept of the window system. Chapter 3 represent the window manipulator which is the tool for man-machine interface of the window system. Chapter 4 describes main classes defined in the window system. Chapter 5 describes the way to use a window from user program, from definition of window class to the deletion of window. Chapter 6 presents the evaluation of window system from the view point of execution speed and man-machine interface. Chapter 7 describes the future plan of window system.

2. Concepts

2.1 Window System

The PSI machine is a Prolog based high level language machine, and a super personal computer. It is developed as the prototype for the fifth generation computer system, and will be used as a software development tool at the second and third stages of the fifth generation project.

As a personal type fifth generation computer system, it is required very high level man-machine interface. The window system is a set of programs which provides the communication pass between the user and PSI machine, and keep the man-machine interface at high level.

The PSI machine is the multi-process machine, and can do many processes concurrently with the request from the user. But PSI machine has only one console, and if the console is connected to a process, the user can communicate with only one process simultaneously. To supply a good interface on the multi process environment, it is needed that user can communicate with some processes concurrently, e.g. user can refer some process simultaneously and can send the operation to any process on machine.

To support these environment, SIMPOS window system supplies some logical terminals named window on the bit-mapped display, keyboard and mouse, and control these windows by the will of the user.

To accomplish these characteristics, the SIMPOS window system works with device handlers, coordinator and each programming systems. Figure 1 shows the structure of systems which work on user communication. The input/output devices are controled by each device handlers. The window system control the basic functions about input/output management (control of window allocation, control of data path from input device to each process, and so on). And the management of inter process communication is managed by coordinator. (Tsuji 84, Kurokawa 84)

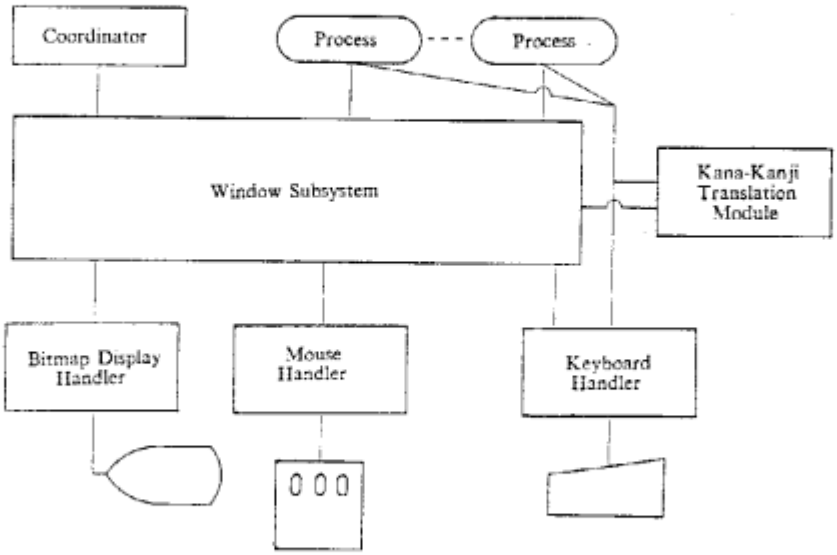

Figure 1. System structure around the Window System

2.2 Window

Each window is one logical terminal for the user. Using some windows, the user can do some tasks at the same time. The Figure 2 shows the PSI's display, and in this situation the user is doing the editing a program and using another window he is debugging the program.

All the programs in SIMPOS are written in ESP (Extended Self-contained Prolog [Chikayama 84]), the prolog based logic program language and it is designed as the Object Oriented language. So each window is defined as a object in the SIMPOS, and each predicate which do the function regard to the window is all defined as the method predicate for that window class.

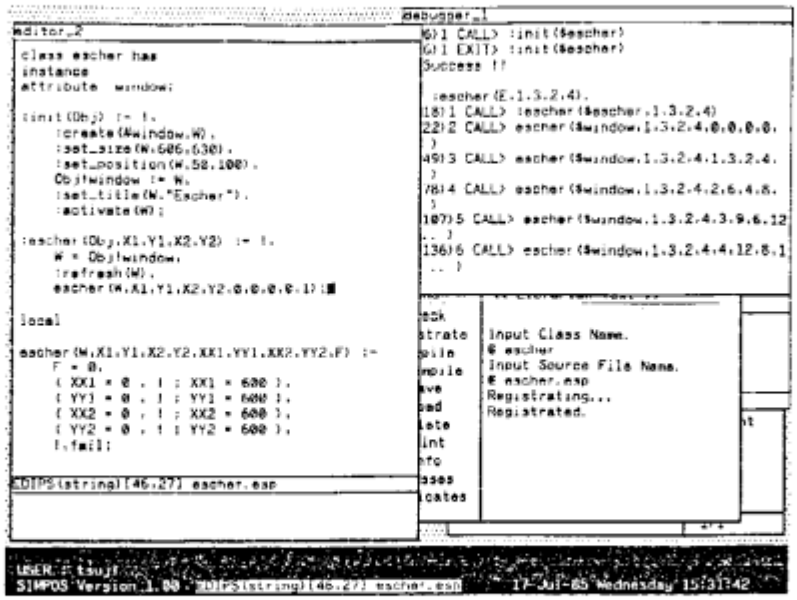

Figure 2. Display image of PSI

2.3 Hierarchy of Window

In the multi-window environment, user sometimes want to use some windows as a set. For example, in the editor user want use the text window and command input window as a set. In this case, it is very useful the editor window has text window and command input window as sub-windows. The SIMPOS window system allows window having some sub-windows. In the SIMPOS window system, the windows consist the hierarchy like tree structure. At the top of this structure, there is a screen, and the ordinary window is defined as the sub-windows of the screen. And each window can have its own sub-windows and so on (Figure 3). In this hierarchy, the management of sub-windows are done by those superior window. The ordinary windows are managed by the screen, and those subwindows are managed by each window, and so on.

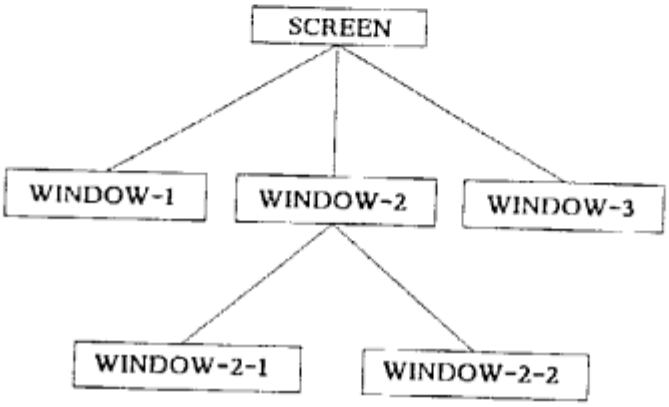

Figure 3. Window Hierarchy

2.4 Input Management

The PSI machine has two kinds of input device, the keyboard and the mouse. These two devices have different feature and they are controled by different way. The keyboard, the text input device is not shared simultaneously by processes, and the input from keyboard must send to the one special window (SELECTED WINDOW). For this reason the selected window must determined by user. The mouse, the pointing device, is the device which is associated to the mouse cursor on the screen, and can walk around the whole screen area. And it can be shared by the windows which are displayed on the screen. The window which the input from mouse is send to is the shown window on which the mouse cursor is positioned at that time.

2.5 Output Management

In the overlaped multi-window environment, there are windows which are partially or fully hidden, and user can not see. When the output is requested to window, the system must recognize the window is fully shown or not. If no check is done, the output may leave not recognized by the user forever.

To control the output to windows, we introduced the statuses of window, exposed/deexposed and shown/unshown.

As mentioned before, windows are managed by its superior window, and the sub-window which is

not hidden by another sub-window has the exposed status. the sub-window which is hidden has deexposed status. The exposed/deexposed statuses represent the status among sub-windows which have same superior window.

The shown/unshown status represents the status whether the window is fully shown on the display or not. The window whose ancestors are all exposed and which is itself exposed has the shown status. And if the window itself or one of its ancestor window has deexposed status, it has unshown status. The window which has shown status is fully shown on the screen.

The most simple way to control the output is to kept waiting the output request for unshown window until it becomes fully shown on the display. But by this way, the emergency information from the process which run on the unshown window can not be shown immediately to the user. It may cause the trouble to that process. There are some other ways to control the output, but they have some disadvantages. And it's not useful to take one way to control.

In the our SIMPOS window system, we take four types together and let the user to select one of them for each window. The four types are as follows:

1) WAIT — to let the window waiting to output until it becomes fully shown on the screen.
2) NOTIFY — to display the message which tells to user that the window want to output through the special window (NOTIFY WINDOW --> to be prepared by system),
3) OUTPUT — to do the output request immediately to the window. If the window is unshown, the output is done to the bitmap area which saves the display image of the window, and only the output to the shown part of the window is displayed on the screen.
4) SHOW — to show the window compulsorily and do the output to the window.

The last two types may cause some trouble. OUTPUT type may leave the information to be not recognized by the user. And when the window which wants to do output is hidden by the selected window, and user is typing the input to the selected window, the SHOW type cause the change of the selected window and send the input from keyboard to the new selected window. So we recommend to user to use normally WAIT and NOTIFY types for safety.

As for the output by kanji character, the PSI machine uses the 16-bit data for character, and kanji character can be treated same as the alphabets. And window system can also treat the kanji character normally, and user has not to conscious the use of kanji character.

2.6 Temporary Window

As mentioned before, the ordinary window is located in the hierarchy of windows, and shown on the screen or unshown by another windows by the request of the user. But in some cases, it's better a window is shown when it is needed and when there is no need to use the window, it is disappeared. For example, some message window is required only when the system want to show some information, and after the user read the message, it is no need to be shown on the screen. SIMPOS window system support this window feature as temporary window. The temporary window is appeared at the position of mouse cursor, and when the mouse cursor goes out from that window, it will disappeared. Temporary window has following features:

1) it never hidden by ordinary window.

2) when it is no need to be displayed, it will not hidden but take off from screen.

3) it saves the area of displaying image where it hides.

2.7 Menu Window

The pointing device mouse is very useful device for pointing some position on the screen, and the most common way to use the mouse is the selection of item in the menu window.

The menu window offer the way of input the commands and so on, by means of find and selection. The user will not be required to remember the commands and he can find the appropriate command among the menu items and complete the operation by selecting the item.

The menu window is the special window which shows some items in it, it accept only the mouse input (pointing and click), and not accept the keyboard input nor output request. The user move the mouse cursor to the item which he want and click the mouse button. By this operation the prepared operation is done for that item.

2.7 Choice Window

The menu window support the function to select one or more items. And when some alternatives are exist for a item, menu require another menu for selecting the alternative. That is to require to the user redundant operation. And in another case, one item have many alternatives like an integer value, the menu cannot manage the selection. In those cases, user want to select one alternative for an item or input the item value from the keyboard. The choice window offers such way for the user. The choice window is keeping items for which the user can change the value. In this window, the item label and the list of alternatives for that item or the current value of that item are shown. Figure 4 shows the sample image of choice window. In this example, a choice window which control the window attributes is shown. The items of SIZE, POSITION, etc. show the value of those items and user can change the value of the item by selecting that item, and typing the value from keyboard. The items like PERMISSION, FONT, etc. shows the currently selected element by reverse display and user can change the element of that item by clicking the mouse button.

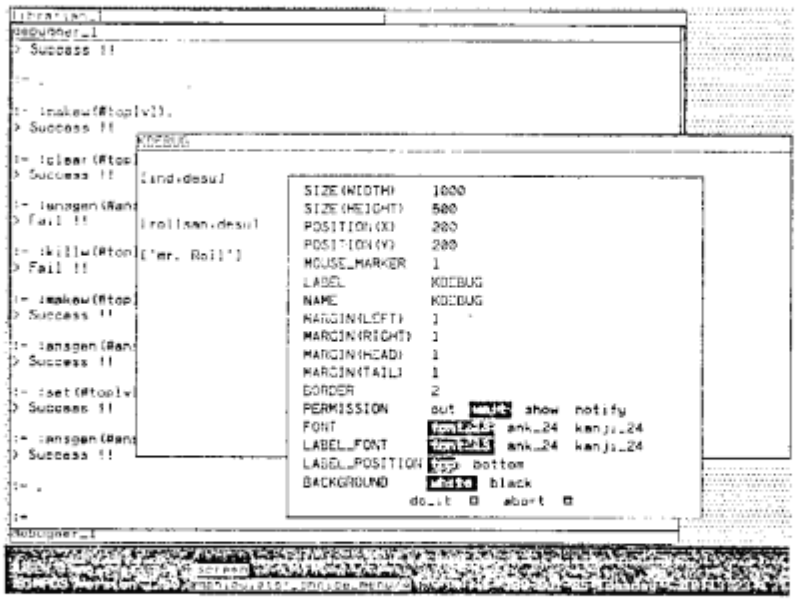

Figure 4. Choice Window

3. Window Manipulator

the window manipulator is the program which change the displaying image of the windows, e.g. size, position, and so on, through the communication with the user. By using this, the user can change the status of window not only by the program but by the mouse, and the user interface around window is kept at high level. When the window manipulator is selected in the system menu, the menu for window manipulator is shown. User select one of the items on the menu and the associated function is invoked. Each item and its function is as follows:

1) SHOW
   Show the whole part of the partially shown window.
2) HIDE
   Put the partially shown window to the bottom of the list of windows.
3) RESHAPE
   Reset the position and size by pointing the top-left corner and the bottom-right corner with mouse.
4) MOVE
   Reset the position of window by mouse.
5) EXPAND
   Expand the window size as large as possible but not hide another fully shown window.
6) MENU
   Select the window using the submenu of windows list.
7) PERMISSION
   Reset the window output permission by sub-menu.
8) ATTRIBUTE
   Change the window's attribute by choice window.
9) HARDCOPY
   Take the window's hardcopy.
10) SUPERIOR
   Change the windows editing base to superior window.
11) INFERIOR
   Change the windows editing base to inferior window.
12) EXIT
   Exit from window manipulator.

4. Classes

The SIMPOS window system is constructed by about 120 classes. This chapter describes main classes. Those classes are classified to three types. One is the stand-alone class which can be used by itself as a window class. Another one is mixin class which supply a function and is used as a super class of another class. And the last one is the internal class which is used by window system itself, and not used by the user.

4.1 Stand-alone Classes

- window

the class for make a standard window

* window_without_label
  the class for making a window which has no labels.
* window_with_two_labels
  the class for making a window with two labels, top label and bottom label. user can use one of the labels for his own usage.
* superior_window
  the class for making the superior window, the window which can have the sub-windows.
* labeled_superior_window
  the class for making the superior window with label.
* menu
  the class for making the standard menu, single select single column menu.
* multi_column_menu
  the class for making the multi column menu.
* temporary_menu
  the class for making the single select single column temporary menu.
* temporary_multi_column_menu
  the class for making the single select multi column temporary menu.
* multiple_select_menu
  the class for making the multi select single column menu.
* multiple_select_multi_column_menu
  the class for making the multi select multi column menu.
* temporary_multiple_select_menu
  the class for making the temporary multi select single column menu.
* temporary_multiple_select_multi_column_menu
  the class for making the temporary multi select multi column menu.
* submenu
  the class for making the sub menu which is the sub window of another window.
* menu_item
  the class for making the menu item. It has the slots, item string, the value to be returned when selected, item type, the documentation for that item.
* choice_window
  the class for making a standard choice window
* choice_item
  the class for making the choice item.
* marker
  the class for making the marker.
* popup_choice_window
  the class for making the choice window with popup feature.
* window_region
  the class for defining the window region.

4.2 Mixin Classes

1) *primitive classes*

* basic_window

the class supplying the function as a window. Every window must inherit this class.

* user_window

  the class for making the standard window which can be used by the user.

* as_inferior

  the class supplying the function as a sub window. All the window class which is defined by the user must inherit this class.

* as_superior

  the class for supplying the function as superior window. If the user want to use the sub windows, The window must inherit this class.

* as_inside

  the class for managing the inside area of the window. The inside area is the part of window which does not include the border, label, and the margins.

* bare_window

  the class for supplying the basic function of the window. For example, making the object, allocating the bitmap area, etc.

* with_border

  the class for supplying the function to have a border of the window.

* with_label

  the class for supplying the function to have a label.

* with_two_labels

  the class for supplying the function to have two labels, top label and bottom label.

* with_margin

  the class for supplying the function to have the margins. The margin is the blank area at the outside of windows.

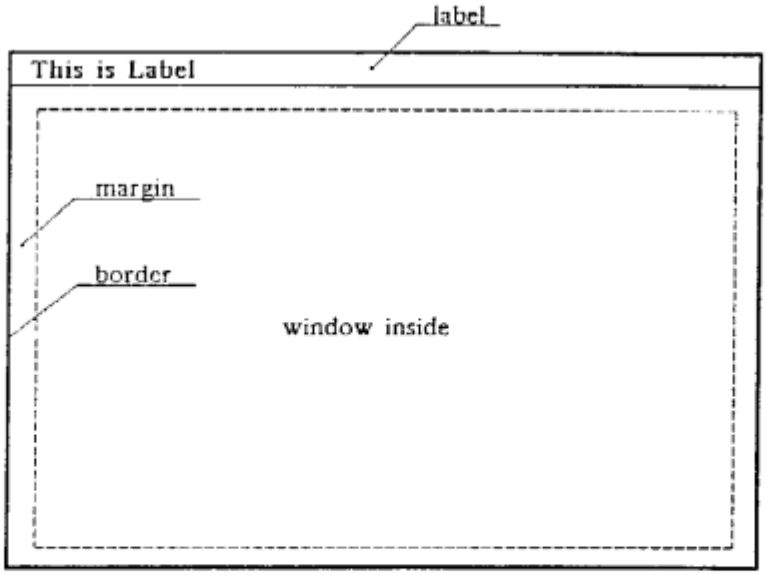

Figure 5. the Border, Label, and Margins

* as_relation

  the class for managing the allocation of the sub windows in the superior window. This class is

inherited by the class as_superior.

* external

  the class for supplying the function to access to the window from outside of window manager. In other words, it supply the function to communication between the user process and window manager process.

* as_temporary_window

  the class for supplying the function as temporary window.

* as_popup_window

  the class for supplying the function as a popup window. Popup window is the temporary window which is not hidden by the exit of mouse cursor.

* sash

  the class for making a window which has border and margins.

* labeled_sash

  the class for making a window which has border, margins and label.

* two_labeled_sash

  the class for making a window which has border, margins and two labels.

2) *output control*

* as_output

  the class for supplying the function of output.

* as_cursor

  the class to manage the cursor which shows the position where the output is done.

* as_string_output

  the class for supplying the function to output a string.

* as_scroll

  the class for supplying the function to scroll the window.

* as_dot_scroll

  the class for supplying the function to scroll the window by dot bases.

* with_mouse_scroll

  the class for supplying the function to cause scrolling by the mouse. this class support the two kind of mouse scroll, by the movement of mouse, and by the click of mouse buttons.

* as_graphics

  the class for supplying the function to output the graphic primitives.

* as_markers

  the class for suppling the function to have markers in the window.

3) *input control*

* as_input

  the class for supplying the function of input from keyboard.

* as_mouse_input

  the class for supplying the function of input from mouse.

* with_default_mouse

  the class for supplying the basic function about mouse control, to detect the mouse movement and to show the mouse cursor in the window, etc.

* with_translation_table

the class for supplying the function to have translation table. The translation table is used for customization and translation the input code to users own character code.

* with_superior_input_buffer
  the class for supplying tne function to share the input buffer with its own superior window.
* with_mouse_scroll
  the class for supplying the function to scroll the contents of the window by mouse.
* with_window_region
  the class for supplying the function to define a mouse sensitive area in that window.
* with_line_selection
  the class for making the window whose each line can be selected by mouse like to item of menu window.
* as_notification
  the class for supplying the function to have the output permission type of "notify". If the window inherits this class and the window output permission is set to "notify", the notify window is appeared to show thuser that the output request is occured, when the output is requested during the window is unshown.

4) *menus / choice windows*

* as single_select
  the class for supplying the function of single select.
* as_multiple_select
  the class for supplying the function of multi select.
* as_select
  the class for supplying the function for selecting the item by mouse.
* as_multiple_column
  the class for making the multi column menu.
* as_menu
  the class for making the menu window. Every menu window must inherit this class.
* as_on off_menu
  the class for making the on-off menu. When the window inherit this class, the item with type of on_off is reversed when selected.
* as_submenu
  the class for supplying the function as a sub menu. In the multi_column menu, each column is defined as sub menu using this class.
* as choice
  the class for supplying the basic function as a choice window. This class manages the choice items defined at this window.

4.3 Internal Classes

* window_manager
  the internal class for window manager process.
* display
  the class for defining the bitmap display. The bitmap display is constructed by the screen and status_line.
* screen

the class for managing the logical screen on the display.

* status_line
  the class for managing the status line. The status line is located at the bottom of the display and it shows some system statuses, date, time, user name, selected window, etc..
* logical_mouse
  the class for the mouse device. This class must have only one object and inherits the class single_object.
* window_message
  the class for making the message for the inter process communication between the user process and window manager process.
* single_object
  the class to limit the number of object of the class.
* window_manipulator
  the class for the window_manipulator. This class control the window manipulator menu.

5. Using the Window

This chapter describes how to use the window system. User can use a window by following sequence.

1) define the window class.
2) create the window object.
3) setup the window attributes.
4) activate the window.
5) operate through the window.
6) kill the window.

5.1 Definition of window class

User can use the stand alone classes which are prepared by window system, or can define his own window inheriting the classes defined by window system.

5.2 Creation of window object

The creation of window object is done by object-oriented call of class predicate, :create.

1) *ordinary window*

:create(Class, ^Window)
  Class => class object of window class.
  Window => returned window object

:create(Class, Initiation, ^Window)
  Class => class object of window class.
  Initiation -> prolog list of items about window initialization, such as window size, position, etc..
    For the items not described at here, the default value is assigned. (Table 1.)
  Window => the returned window object.

Example:

:create(#window, [size(200,400), position(100,200)], Window),
create the standard window with size of 200 x 400 and position of (100,200).

Table 1. Window Initial Options

Initiation	Remarks	Default Value
superior(Superior)	the superior window	screen
position(X,Y)	window position in the superior window	(0,0)
size(Width,Height)	window size	inside size of superior window
title(String)	the label title	blank
top_title(String)	the top_label title	blank
bottom_title(String)	the bottom_label title	blank
margins(Left,Right, Head,TAil)	the margins	1 dot
units(Width,Height)	the inside units (the grid size in the window)	1 dot

2) *menu window*

The creation of menu window is almost same as ordinary window, but the initiation list must include the description about menu items, items_list(Menu_items_list). Menu_items_list is the prolog list of vectors which describe the each items specification. And each vector must keep the following form.

{ item_string, item_type, item_id, documentation }
item_string => string displayed in the menu.
item_type => type of the item.
item_id => the value which will be returned when that item is selected.
documentation => the guide documentation displayed at the status line.

3) *choice window*

The creation of choice window is almost same as ordinary window, but the initiation list must contain the description about the choice items, choice_list(Choice_items_list). Choice_items_list is the prolog list of the vectors which show each items specification. And each vector must keep the following form.

{ item_string, item_type, elements_list, initial_element, documentation }
item_string => strings displayed in the choice window.
item_type => type of the item.
elements_list => prolog list of alternatives of that item.
initial_element => initial value of the item.
documentation => guide displayed on the status line.

5.3 Setting the attributes of the window

Setting the attributes like size, position, superior window, etc.. Most of these attributes can be setted by the initiation list at the creation.

1) *Basic attributes*

:set_size(Window, Width, Height)
set the size of window.

:set_position(Window, X, Y)
set the position to the superior window.

:set_superior(Window, Superior_window)
set the superior window.

2) *Add/delete the menu item*

:add_item(Menu, Item, Position)
add a new menu item to the single column menu at the specified position. Item is the vector which specify the item and its form is same as described at the menu creation. Position is the integer between 0 and the number of items.

:add_item(Menu, Item, Column, Position)
add a new menu item to the multi column menu at the specified column and specifyed position.

:delete_item(Menu,Item_string)
delete the item which item_string matches to the Item_string from the menu window.

3) *Border/Label/Margins*

:set border_width(Window, Width)
set the width of border by dots. The default value is 2 dots.

:set_label_position(Window, Position)
set the postion of label (top or bottom). Position is an atom "top" or "bottom".

:set_title(Window, String)
:set_top_title(Window, String)
:set_bottom_title(Window, String)
set the string displayed as the label.

:set_margins(Window, Left, Right, Head, Tail)
set the width of each margins by dots.

5.4 *Activation of the window*

:activate(Window)
activate the window. When activated, the window is scheduled to be displayed on the superior window.

:show(Window)
show the window on the screen.

:deactivate(Window)
deactivate the window, e.g. delete the window from scheduling on the superior window. This predicate don't kill the window, so, the displaying image is kept as it is displayed.

:hide(Window)
hide the window on the superior window. The part where no other window is exist will be kept as displayed.

5.5 Output

1) *text output*

:write(Window, Character)
output the Character to the cursor position and put the cursor one character forward.

:move_cursor(Window, X, Y)
move the cursor to the position of (X,Y).

:insert(Window, Character)
insert Character to the cursor position.

:delete(Window, Character)
delete the Character from the cursor position. By this predicate, the position of the cursor is not changed.

:clear_line(Window)
delete the characters between the cursor position and the end of line.

:scroll(Window, Offset)
scroll the window contents by Offset. If the Offset is positive integer, scrolling up will be done, and if the Offset is negative integer, scrolling down will be done.

:open_rows(Window, Row, Offset)
insert the Offset lines at the Row position.

:move_rows(Window, Row1, Row2, Offset)

move the lines between Row1 and Row2 by the value of Offset.

2) *Graphic Output*

:get_point_value(Window, X, Y, ^Value)
:set_point_value(Window, X, Y, Value)
read/write the value (0/1) of the pixel specified by (X,Y).

:draw_line(Window, X1, Y1, X2, Y2)
:draw_line(Window, X1, X2, X2, Y2, Line_width, Line_type, Alu)
draw a line between the point (X1,Y1) and (X2,Y2).
Line_width => width of the line (default is 1)
Line_type => type of the line (default is fixed line)
Alu => boolean operation for drawing (default is exclusive or)

:draw_rectangle(Window, X, Y, Width, Hight)
:draw_rectangle(Window, X, Y, Width, Hight, Line_width, Line_type, Alu)
draw a rectangle at the position of (X, Y) with test size of (Width, Hight). The Line_width,Line_type and Alu are same as :draw_line.

:draw_string(Window, X, Y, String)
:draw_string(Window, X, Y, String, Alu)
draw the string "String" at the position of (X, Y). the position specifies the top_left position of the first letter of String.

3) *markers*

:create_marker(Window, ^Marker)
create the marker object.

:draw_marker(Window, Marker)
display Marker on the window.

:erase_marker(Window, Marker)
erase Marker from the window.

:move_marker(Window, Marker, X, Y)
display the marker at the position of (X, Y).

:set_marker_type(Window, Marker, Type)
set the marker type (box or line).

:set_marker_size(Window, Marker, Width, Hight)
set the marker size.

:get_marker_list(Window, ^Markers_list)
return the list of all markers the window have.

5.6 Input

:read(Window, ^Code)

read a character (or mouse click, term, etc.). If there is no input, the process will be kept waiting.

:read_sense(Window, ^Code)

almost same as :read, but if there is no input, this predicate returns the atom "$$nil".

In the SIMPOS window system, above two predicates are main input predicate. The returned code will be different by whether the input is done by keyboard/mouse or by what class the window inherits.

1) *Input from keyboard*

ordinary, the character code will be returned.

2) *Input from mouse*

mouse_click_code will be returned. The mouse_click_code is coded on the source codes like:

mouse$r -> mouse right button single click
mouse$m => mouse middle button single click
mouse$l => mouse left button single click
mouse$rr => mouse right button double click

3) *Input with translation table*

when the window inherits the class "with_translation_table" the input from keyboard and mouse is translated by the translation table and associated code will be returned as result.

4) *Input as mouse_scroll*

when the window inherits the class "with_mouse_scroll", the user can require the scroll by mouse. When this request is done, the term "scroll_request(Offset)" will be returned as code.

5) *Input by menu selection*

If the window is a menu, by the selection of menu item the item_id associated to that item will be returned as code. If the menu is a temporary menu and the mouse exit from the menu, the atom "abort" will be returned.

5.7 Deletion of window

Delete the window from the system when the user terminate his job on that window and he want to delete it.

:kill(Window)

delete the window from system and release the bitmap area which the window occupied. The window is registered to the process which create the window, and when user killed that process, all windows that process have will killed automatically.

5.8 Sample codings

Here, We show the sample program which create a window with size of 300 X 200 and position of (100, 50), draw line and output a character input from keyboard.

```
:sample( Obj )  :-
                .
                .
                .
    % create window
   :create( #window, [ position( 100, 50 ), size( 300, 200 ) ], Window ),
    % activate window
   :activate( Window ),
    % draw line
   :draw_line( Window, 10, 50, 250, 150 ),
    % input character
   :read( Window, Character ),
    % output the character
   :write( Window, Character ),
                .
                .
                .
```

6. Evaluation

When user uses the PSI machine, he directly access to the window system, and it is very important to the man-machine interface at the execution speed and its easiness to use. In this section, we describe the estimation about those of our window system.

6.1 execution speed

Our SIMPOS project is the first challenge to write an operating system by the prolog-based logic programming language, we try to make a window system without considering the execution speed so well in the early stage. It began to work on July of 1984, and it takes about one year and half from starting the detail design. But it was too slow and we started the revisision of window system to make more efficient one. We change the configuration of window system and device handlers, and we recoding the many part of window system.

1) *output*

For the measurement of one character output,

```
:write( Window, Character )
```

we use the window with following conditions.

* window classes

ordinary window which inherits the following classes.

labeled_sash, as_scroll, as_output, as_graphics, as_markers, as_input, as_mouse_input, user_window

* window status

shown on the display.
without cursor.
output position not causing the scroll nor wrap around.
output character is printable character.
font size is 13 dots wide, and 19 dots hight.

* measuring points

1. making window message and process change
2. analysis of window message and calculation of cursor position
3. deletion of cursor
4. calculation of parameters of character displaying.
5. character displaying
6. calculation of new position of cursor
7. displaying the cursor
8. making the reply message
9. process change and analysis of reply message

* revision items

a. first version
b. poling version of bitmap display handler
c. subroutine version of bitmap display handler
d. revision of inter process communication
e. firmware support of method call and slots access

* result

Table 2. Result of estimation of output

msec.

		a	b	c	d	e
1		-	-	54	27	3
2		18	18	13	13	2
3		216	146	5	6	2
4		22	22	9	10	2
5		333	193	10	12	5
6		23	23	12	11	1
7		257	146	5	6	2
8		8	8	3	2	1
9		-	-	65	36	3
total	I	922	556	57	60	15
	II	-	-	171	123	21

I : total time in window manager process.
II : total time of message execution.

* evaluation

From the first version to firmware support, it cost about four months, and we think it very rapid improvement. Of course, it's not fast enough and we must improve window system more and more. But it can be used for software development tool because the PSI machine is personal machine and only one user can use its CPU time.

2) *input*

For the measurement of one character input,

:read(Window, Character)

we use the window with following conditions.

* window classes

same as evaluation of one character output.

* window status

shown on the display.

without cursor.

called the :read predicate and waiting the keyboard input.

* measuring points

1. processing in keyboard handler
2. process change
3. processing in window manager
4. process change
5. processing in user process

* revision items

a. first version

b. revision of process change

c. revision of keyboard handler processing

d. direct connection of keyboard handler and user process

e. revision of process change

f. firm ware support

* result

Table 3. Result of estimation of input

msec.

	a	b	c	d	e	f
1	96	54	36	35	24	4
2	33	20	20	20	15	2
3	15	15	15	-	-	-
4	32	24	24	-	-	-
5	3	5	5	4	5	1
total	179	118	100	59	44	7

• estimation

Now it takes about 7 mil-seconds to input one character. It can not be said to be enough, but it takes about 4 mil-second in the keyboard handler. Processing in the device handler is deterministic one. Now the firm ware group will planning to introduce if_then_else notation and other os support, and by that revision the processing in the device handler will become more faster.

6.2 man-machine interface

When considering the man-machine interface, the easiness to use is most important problem. In the multi-window environment, user can refer many informations from various system programs, e.g. the program source by the editor, error message from compilor, trace information from debugger, etc.. its very useful to develop a program on the computer. And in such environment, there are some problem about man-machine interface.

1) *easy to select a window*

In the multi-window environment, the user work around the windows during his work. In the overlapped type window system, some windows may be fully hidden and user may go to such window to do some job. In those cases, he want to go another window easily, and system must supply simple way to go to another window easily.

Our window system supply the two ways for this purpose. One is selection by mouse button. User can select the partially or fully shown window by clicking the mouse left button on the window he want to select. The other is selection by menu. User can select even the fully hidden window using the window manipulator's menu (item of select by menu). These ways are very useful for using many windows on his job. Of course the user can go to another window by program using the :select(Window) predicate.

2) *easy to edit the allocation of windows*

The multi window environment will offer the capability to refer some windows simultaneously like the situation of to work on the desk spreading some papers on it. But as you know, man is often change the allocation of papers on the desk to refer some papers. In the multi window environment, if it's hard to change the windows allocation, it's rather hard to refer windows each other.

Our window system has the tool, "window manipulator". And it offer functions to change the allocation of windows using the menu and mouse. User can show, hide, move, or reshape windows very easily, and he can easily refer another window by using this function.

3) *easy to define the window which have some feature user want*

When user use the window, he may want various features about it. For example the menu window is useful for find and selection of command input, but for some items user will want many alternatives or want to set the value to the item by keyboard input like integer number. If the window system will offer standard window only, it seems not good by its flexibility, and it seems very good to offer the capability to define the window features by user's will.

Our window system is written by ESP, and by fully utilizing it's object oriented features user can

define his own windows by inheriting the component classes to his own window. For example, Figure 6 shows the display of debugger system's state window which control the trace modes of the debugger. It has four sub-windows. The top window is the window controlling the mode of debugging and representation level of trace information. The item of mode selection is selection of three alternatives, and representation level (Depth and Length) is setted by keyboard input. The second window is to control the timing to show the trace informations, and it has two dimensional items because the items are so many. The third window is control the selection of kinds of predicates to show the trace informations, and it has items which is located horizontally by the relation with another subwindows. The bottom window is control the spy point, and spy point is setted by keyboard input. These sub-windows are defined by debugger itself using the system support component classes.

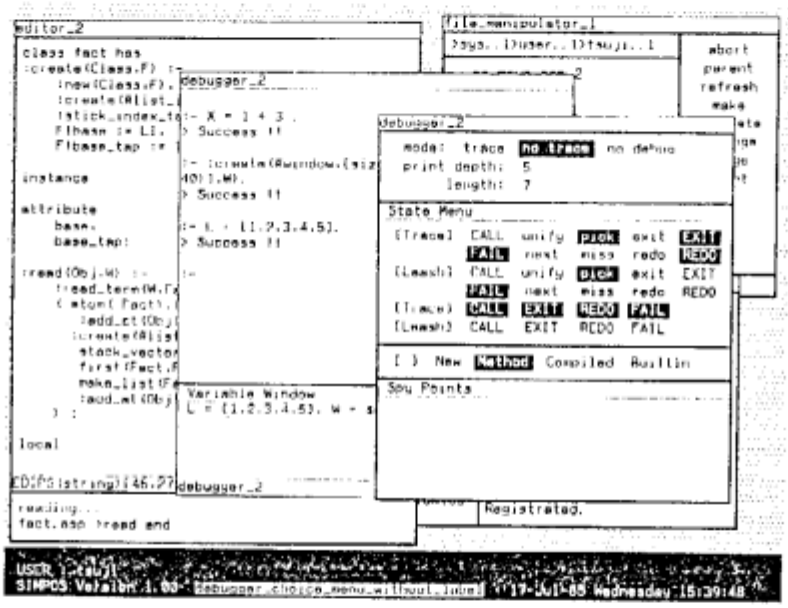

Figure 6. Debugger system's state window

4) *easy to control the window from program*

Easiness to use the window by user is very important, but the program must use the window for input/output, and if it's very hard to use the window from program, the program cannot fully utilize the easiness of window. The SIMPOS window system is written by ESP, and each window is defined as an object. And many useful methods are prepared for the program. The system programmer can use those methods, and can easily access to the each window.

5) *easy to define the structure of the windows*

Some program like debugger require some windows for output many kinds of information and input user command input. In that case, the system programmer don't want to control those windows individually. In the SIMPOS window system, there are some classes for making the structure of windows and can define some windows as subwindows of one window. So the user can build his own structure of windows easily and without detailed control he can manage the windows.

7. Future Plans

Nowadays, about 30 PSI machines are released to the members of FGCS project, and SIMPOS version 1.0 is working on those machines. The window system is also working on these machine, and used to the some project in the ICOT. But the window system and SIMPOS working is not concluded and it will be revised to be more friendly machine.

1) *Multi-font*

Now only two font, Alpha-numeric font and kanji font are released and used on the PSI machine. But, recently the prototype of font editor is made and many font will be designed for some program. Now SIMPOS window system supports the multi font environment, but its not so easy to use the multi font on one window. We will support the multi font environment including the variable pitch font.

2) *Standard input/output*

In the SIMPOS, there are three medium-subsystems, window system, file system, network system. they are developed independently and their interface are not standardized. User or system programmer must use those systems individually and must notify which medium to use as input/output. We are planning to standardize those medium systems and to offer standard input/output to user.

3) *Automatic configuration of subwindows*

In the SIMPOS window system, we are supporting the function for making the window structure, but now, user must consider the allocation of subwindows in the superior window. So when the user want to reshape the superior window, the program must reallocate its subwindows according to the new superior window's size. We are planning to support the automatic configuration of subwindows to reallocate the subwindows automatically at the time of superior windows reshaping. The user or system programmer can specify the percentage wise size and position of subwindows in the superior window and when the superior window is reshaped, the system calculate the new size and new position of subwindows automatically.

The word "SIMPOS" means progress in japanese, and SIMPOS will improved continuously. The window system will improved by the reaction from users, and will become more useful and easy system.

Acknowledgement

The author express their grateful thanks to Mr. Kazuhiro Fuchi, Director of ICOT Research Center, and Dr. Toshio Yokoi, Chief of third laboratory for their continuous encouragement, and to other members of all the researchers of ICOT 3rd. laboratory for their advice and discussion about development of window system. The author also thanks to Members of SIMPOS Window Sytem Group (Mr. Yutaka Iima, Mr. Osamu Nakazawa, and Mr. Shoji Enomoto) for their works on window system development.

References

[Chikayama 84] Chikayama, T., "ESP Reference Manual", TR-044 1984

[Chikayama 84] Chikayama, T., "Unique Features of ESP", TM-0055 1984 (Also in "Proceedings of FGCS'84", Tokyo, 1984)

[Hattori 84] Hattori, T., Tsuji, J., Yokoi, T., "SIMPOS: An Operating System for a Personal Prolog Machine PSI", TR-055 1984

[Kurokawa 84] Kurokawa, T., Tojo, S., "Coordinator - the Kernel of the Programming System for the Personal Sequential Inference Machine (PSI)" TR-061 1984

[Takagi 84] Takagi, S., et al., "Overall Design of SIMPOS (Sequential Inference Machine Programming and Operating System", TR-057 1984 (Also in "Proceedings of 2nd Int'l Conference of Logic Programming", Uppsala, Sweden, 1984.)

[Taki 84] Taki, K., et al., "Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)" TR-075 1984 (Also in "Proceedings of FGCS'84", Tokyo, Japan, 1984.)

[Tsuji 84] Tsuji, J., et al., "Dialogue Management of the Personal Sequential Inference Machine (PSI)", TR-046 1984 (Also in Proceedings of ACM'84 1984)

[Yokota 84] Yokota, M., et al., " The Design and Implementation of a Personal Sequential Inference Machine: PSI", TR-045 1984 (Also in New Generation Computing, Vol.1 No.2, 1984)