TR#90-004

An Overview of *Yy* and *YyonX*

– A CLOS based Window Tool Kit and its Implementation –

Technical Report No. 90-004

Masayuki Ida[*], Keisuke Tanaka[†],
Takashi Kosaka[‡], and Yukio Ohta[§]

Computer Science Research Laboratory
Information Science Research Center

Aoyama Gakuin University
Address: 4-4-25 Shibuya, Shibuya-ku, Tokyo Japan 150

Feburary 25, 1991

[*] ida@csrl.aoyama.ac.jp
[†] keisuke@csrl.aoyama.ac.jp
[‡] kosaka@csrl.aoyama.ac.jp
[§] yohta@csrl.aoyama.ac.jp

Preface

This report consists of three parts: Part I is the paper for *Yy* and *YyonX*. The early version of it was appeared in the proceedings of Europal'90 held at Cambridge UK, pp 245 - 252 (March 1990). Part II is a quick introduction for *Yy* and *YyonX*. Part III has two sections. One is the contents of *YyonX* version 1.2 release note which was set to the anonymous FTP machine named `ftp@csrl.aoyama.ac.jp`. The other is the example of the *Yy* application. A source code for 'grapher' and its explanation are included.

Masayuki Ida is the project leader/designer and the principal author of the documents. Keisuke Tanaka is the principal author of *Yy-server.* Takashi Kosaka is the principal author of *Yy-client.* Other contributors of *Yy-client* source codes are Yukio Ohta and Masayuki Ida. Symbolics specific extensions are contributed by Eiji Shiota. Yukio Ohta played the role for debugging, application development, and image/animation features. The basic documents for image/animation features were written by Yukio Ohta.

With the Cooperation of The *Yy* Research Group of the year 1990

Members:
Masayuki Ida (Aoyama Gakuin University),
Keisuke Tanaka (Aoyama Gakuin University),
Takashi Kosaka (Aoyama Gakuin University / CSK Corp.),
Yukio Ohta (Aoyama Gakuin University / CEC Ltd.),
Katsuhiko Yuura (Hitachi Ltd.),
Eiji Shiota (Nihon Symbolics Corp.),
Haruyuki Kawabe (Nihon Unisys Ltd.),
Atsushi Atarashi (NEC Corp.),
Noritoshi Rokujo (Fujitsu Ltd.),
Mamoru Sato (Telmatique International Research Lab.)

— No Migration but Co-existence —

Contents

— No Migration but Co-existence —

Part I
Yy and YyonX

Abstract

Yy is a portable window tool kit on top of Common Lisp. Yy was originated in the needs for a common window system.

Yy is outlined as 1) a three level layered model; YYAPI (application interface), YYWS (window system) and NWSI (Native window system interface). 2) a window tool kit or UIMS on top of other general window system: Yy is not a competitive new window system. 3) a CLOS oriented system for both internal architecture and user interface. 4) a presentation system and an output recording facility.

YyonX is an implementation of Yy on X-window. The pilot of YyonX is already working. It uses a server/client model. The protocol is defined for communication between a server and a client. Yy-server is a X-client for NWSI and low level stuffs of YyWS. Yy-client is a X-client for high level stuffs of YyWS and YYAPI. In the paper the design of basic classes and generic functions for Yy are also introduced.

1 Background 1 : Needs for a Window System in Common Lisp

There are several Window Systems and UIMS for various Common Lisp implementations. But they have different functionalities. With this reason, though the portability of Common Lisp is proven, lots of interactive and/or graphical applications can not be ported to different implementations easily.

From 1984 on, there have been several Common Lisp based standardization efforts including window systems. In the beginning, the Common Windows of Intellicorp was one of the candidate for standard but was not adopted. At that time, Symbolics Lisp machine did provide the Dynamic Window which features the presentation system. Common Windows itself grew and several dialects of it were introduced.

We made surveys and got a sketch as follows:

1) Trial to integrate several existing window systems into one is required. It should be independent from any existing windows but should have a bridge from them.

2) Consultation with the current technology is required. At least, X-window and it's friends, Common Windows, Genera, and PC-based windows should be investigated.

3) Public acceptance of the superiority of Dynamic Windows feature should be checked.

4) Good environment needs large memory. Is it affordable ? Say, a full function programming environment may occupy 20 M bytes or more.

5) As a workstation, much more functionality on Japanese character handling in every

text handling is required.

2 Background 2 : CLOS and Object Oriented Approach

Recently it is quite common to integrate a windowing stuffs using object oriented idea. This trend has a firm technical background. Smalltalk is one example. In the Lisp world, Flavors is used as a kernel technology for Symbolics Genera Window system.

Recent development of CLOS (Common Lisp Object System), and its adoption as a part of X3J13 Common Lisp, place Common Lisp among the family of object oriented languages. Since CLOS (chapter 1 and 2) was established in 1988, there are not so much experience reported yet.

Lots of existing window systems for Common Lisp lack of object oriented approach since they were born before CLOS was invented. To CLOSify a portable window system is a must.

3 Analysis of Requirement Issues

Issue 1. Single Process or Multi Process

▷ discussions: Related questions are whether each window is assigned an independent process or is assigned to a window of native window system or no.

▷ conclusions:

1) Place a root window at initialization. Each window is a child of it.
2) Each window can be assigned a process.
3) Keyboard and Mouse events should be correctly and promptly handled.
4) "Pure Multi Process" is not needed. (Each window is not needed to be independently executed)

Issue 2. Object-Oriented Style

▷ discussions: CLOS is a must from the background requirement.

▷ conclusion:

1) User interface and window handling should be CLOS based both.
2) Window can be dynamically defined.
3) Try to evaluate the CLOS power. Try to check whether Meta Object Protocol give us a

solution or not.
4) Migration path from other object oriented windowing tools is needed.

Issue 3. What is the Displayed Output.

▷ discussions: We discussed about what is the major advantage of Genera window system. One thing to prove is whether presentation system is affordable. In Genera, output is recorded as much memory as exists. (user can clear the history freely). Is infinite recording really necessary considering the space consumption and redisplay speed. Should all the object be mouse sensitive items? Context dependent mouse sensitivity is affordable or no.

▷ conclusions:

1) Must install output recording mechanism. Should provide a constant which has the maximum numbers of lines/items. (we all agree with the output recording feature is useful though it is heavy. need some purging mechanism.)
2) Presentation type is very important. Prepare the facility to get user defined presentation type. A new definition using CLOS named presentation class is introduced.
3) Sensitivity control considering the mouse speed should be needed.

Issue 4. Font and Multi National Character Presentation

▷ discussions: Japanese characters should be handled. Can display several fonts in the window or no. Two things to consider are, a case for multi-font strings and a case for alignment / pitch control of displayed characters with variable fonts.

▷ conclusion:

1) Provide multi font ability, though font itself is implementation dependent. The mechanism for it can be standardized.
2) Must consider the provision for dynamic adjustment of displayed characters with various fonts.

Issue 5. Widgets or Accessories

▷ discussions: Lots of window systems have lots of widgets/gadgets and independent styles. User friendliness comes from a uniformity of operations.

▷ conclusion:

1) Too much consideration for style guiding is not needed.
2) Coordination with Existing Styles or the styles of native window systems is needed.

Issue 6. Input Editor

▷ discussions: We discussed about the needs for input front-end processor (in a broader sense), Input Editing and Japanese character input mechanism as an example. There exist several Japanese character front-ends. Wnn, egg, or commercial front-end package, should co-exist with our new window at least with flexibility.

▷ conclusion: There must be a common way to access the stream internal buffer to alternate the character already input to that input stream.

Issue 7. Graphics Functions

▷ discussions: Provide rich functionality ? 3D model is needed ? Color handling is needed ?

▷ conclusion: The Lisp window system we are thinking doesn't provide high level graphic tools, but a typical level ones. Must provide color capability.

Issue 8. Implementation Model

▷ discussions: Is network oriented implementation needed ?

▷ conclusion: We don't need to specify the model details, since the target machines have much variety.

Issue 9. A New Window System or a Tool Kit on Top of other Windows

▷ discussions: Creating a new window system is attractive solution for the above. On the other hand, if we start to develop a new window system from scratch, we might need to develop every codes which should cope with the same functionality as other windows will have in every minutes.

We want to share the progress of window technology as a user of other window systems.

▷ conclusion: We will design and provide a window tool kit or user interface management system on top of various general purpose window systems.

4 The Design of *Yy Window Tool Kit* as a Conclusion of the above Analysis

Yy Window Tool Kit is the goal of our analysis and is an output of our research works. As a summary, *Yy* is a CLOS based window tool kit with our scheme of output recording. We design the *Yy* as a layered tool kit with three levels; NWSI, *YyWS*, and *YyAPI*.

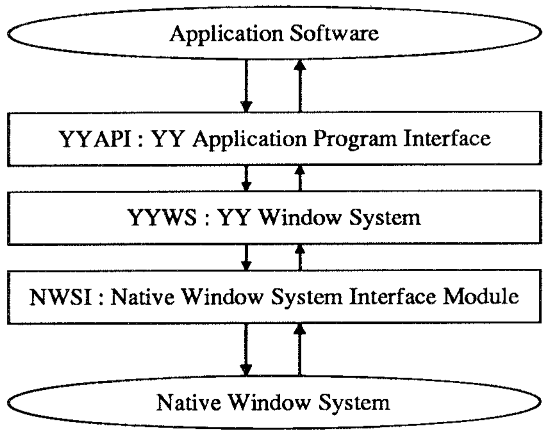

Application Software
YYAPI : YY Application Program Interface
YYWS : YY Window System
NWSI : Native Window System Interface Module
Native Window System

Figure 1: ***Yy*** Layered Model

NWSI: Native Window System Interface Module
YYWS: *Yy* window system
YYAPI: *Yy* Application Program Interface
Figure 1 shows the model. NWSI depends on a native window system. *YyWS* is a native window independent window system and manages all the window objects. The interface between *YyWS* and NWSI is defined to be independent from various underlying window systems as possible. *YyAPI* (Ida, 1989) is an Application Program Interface and is used by the users.

5 *YyonX* as an Implementation of ***Yy Window Tool Kit***

5.1 The Server-Client Model of *YyonX*

YyonX is a ***Yy Window Tool Kit*** on top of X-window, and is implemented as a pilot.

The three layers of ***Yy Window Tool Kit*** are implemented by server-client model shown in Fig.2. *Yy-server* is a server for ***Yy Window Tool Kit*** . *Yy-client* is a client. All the window operations are done by the cooperation of these two process.

Yy-server has two parts; The device dependent part and the device independent part. The device dependent part is a NWSI for X-window. The device independent part is a kernel of *YyWS*.

Yy-client has two parts; the application part of *YyWS* and the YYAPI.

Yy-server and *Yy-client* are implemented as X-clients. Then, they can be loaded on separate machines on the network. There is *Yy* protocol defined in (YY Project, 1989).

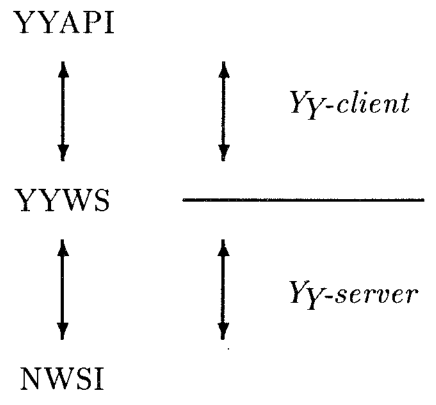


Figure 2: Implementation of *Yy* with a server-client model

5.2 Running Environment

To run *YyonX* the following requirements should be satisfied.

1) Full Common Lisp, 2) X window R11, 3) CLOS implementation (as a pilot implementation, use PCL)

6 Characteristic concepts of *YyonX*

6.1 Meta Coordinate system

The coordinate system of X window has the origin at the top-left corner. While, many graphics systems have the origin at the bottom-left corner. To provide a flexibility on selecting the coordinate system in application software, user can choose which origin is suitable at the time of window creation. Since this feature uses the inheritance switching, all the selections are done at the initialization stage, there is no overhead on run time.

6.2 Output Recording and its Dynamic Control, Presentation

YyonX has an output recording and a presentation system. The basic components of presentation system are implemented as CLOS classes. Our presentation system is simpler than Symbolics' one. There is no implicit inheritance. The maximum number of recorded objects is controlled by a special constant.

Presentation system provides a mechanism to accept an output recorded object on the screen rather than typing the same thing. Since ***Yy Window Tool Kit*** has a mouse-still concept at the kernel level and is used to trigger an inspection procedure asynchronously, placing a mouse to a candidate for a certain short period triggers to check the class and place a wire rectangle around it if it is a successful candidate. If it is the user's choice, clicking it accepts it as an object to input.

6.3 Functionality level which is equivalent to Common Windows

As many feasible drawing primitives as possible will be provided. The functionality level of *YyonX* application programmer interface is arranged as an equivalent to Common Windows.

6.4 Input Editor and Multi-National Features

One of the problem is which font is assumed to display multi-national characters. As for japanese characters, there are several ways to provide japanese character fonts. The font set for *YyonX* is selectable.

As for Japanese text input front-end, *YyonX* can switch between a custom made front-end and a Wnn Jserver.

Mini buffer is provided for input editing.

As for character font manipulation, character data type of Common Lisp has an attribute for it, and we will follow the update compiled by X3J13.

6.5 Distribution of the Loads

Server/Client model enables a distribution of the CPU and memory loads into two machines. Fig.3 shows a case for three machines' co-operation. Machine A and Machine B communicates with X-protocol. Machine B and Machine C communicates with *Yy*-protocol. With this configuration, actual application is supposed to be on machine C, and Japanese input front-end which sometimes needs heavy loads on CPU and memory can be separated from actual application on C.

Yy-protocol uses 4 byte unit packets to communicate. It includes all the functions with compact representation. Currently we have 48 instructions and three types. Intermittent synchronization feature enables asynchronous operations of the both sides. For minimum case, only 12 bytes are necessary to command. *Yy* protocol is designed to reduce the network traffic without degrading the functionality.

7 Primitive Classes

7.1 Position and Region

Position and Region are the most primitive classes in *YyonX* . Position is an object for a position in XY coordinates. Region is an object for a rectangular area.

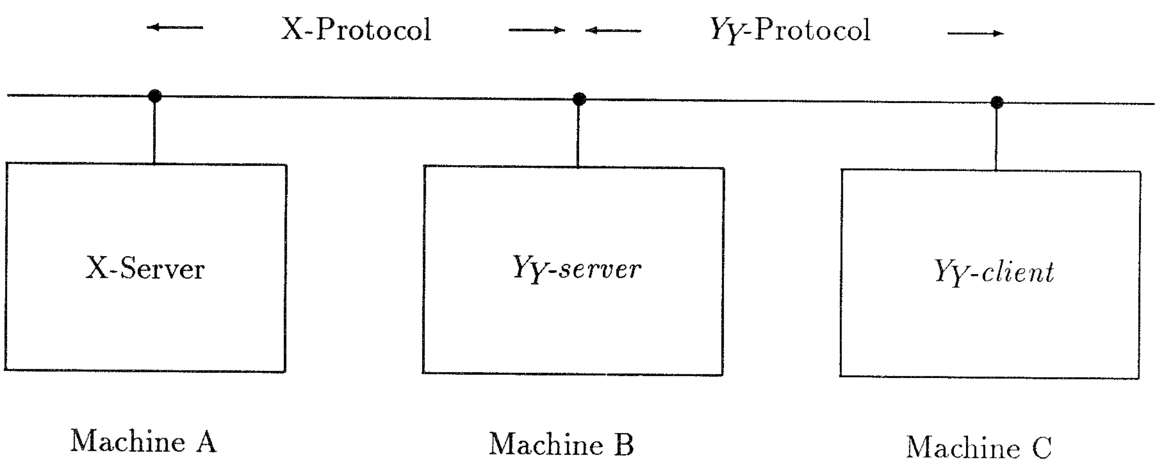


Figure 3: A Maximum Case for *Yy* Execution Environment

Region has six slots logically; :top, :bottom, :left, :width, :height, and :right.

Region can be handled with LBRT (left-bottom-right-top) concept or LBWH (left-bottom-width-height) concept as user likes it.

7.2 Active-Region

Active-region is mouse-sensitive region. Several mouse-methods like enter-region, exit-region, are provided.

This active-region mechanism is completely independent and different from presentation. With active-region, user may define arbitrary rectangle.

7.3 Stream

Input and output with ***Yy Window Tool Kit*** are through the streams. The implementation is done with CLOS, and is an extension of stream type conforming David Gray's proposal to X3J13.

The primary streams are a `graphic-stream` and its two sub classes named `window-stream` and `bitmap-stream`.

On the other hand, `event-stream` is provided to support the handling of asynchronous input events. Mouse cursor movement, button clicking information, event status, and keyboard interruption are passed via event-stream.

7.4 World, Viewport and Page

The region which is a subject to display is called a world. The part of world which is appeared on the screen is called viewport. Page is for a fixed width world.

7.5 Stipple

The target of `bitmap-stream` is called stipple. Object can be transferred between stipple and world.

7.6 Class management in *YyWS*

In *YyWS*, all the windows, window components and presented outputs belong to classes. These classes have several multiple inheritance relations.

Region class is used as one of the most primitive classes and is inherited by several window classes. As its nature, class inheritance is used to provide a module/procedure library.

8 Window Stuffs

8.1 The Structure and the Operations for Window

Window is composed of title and frame. Title might not exist. Accessories can be attached to title and frame. A frame is a viewport or a page. The edge of frame can have a width and is called border. Frame supports scrolling. Viewport scrolling is possible for top-bottom and left-right. Page scrolling is only possible only for top-bottom. Fig.4 shows the relation between world, viewport and window. Coordinate system indicator is for the meta coordinate system. The indicator of Fig.4 example shows the origin is the left-bottom.

Window class definition has slots to describe parent relationship and sibling relationship. The root is a root window.

Window has a rectanglar area and is composed of title-bar, border, scroll bar, coordinate system indicator area, and frame. Viewport or page is mapped to frame.

Both graphic drawing and text display are permitted on viewport. Top-bottom and left-right scrolling are allowed on viewport.

On the otherhand, Left-right scrolling is not permitted for page. The usage of page is assumed as interaction buffer like a lisp listener or a emacs editor buffer which can be defined as a fixed width roll paper. To achieve high speed display on page, graphic drawing on page is not considered.

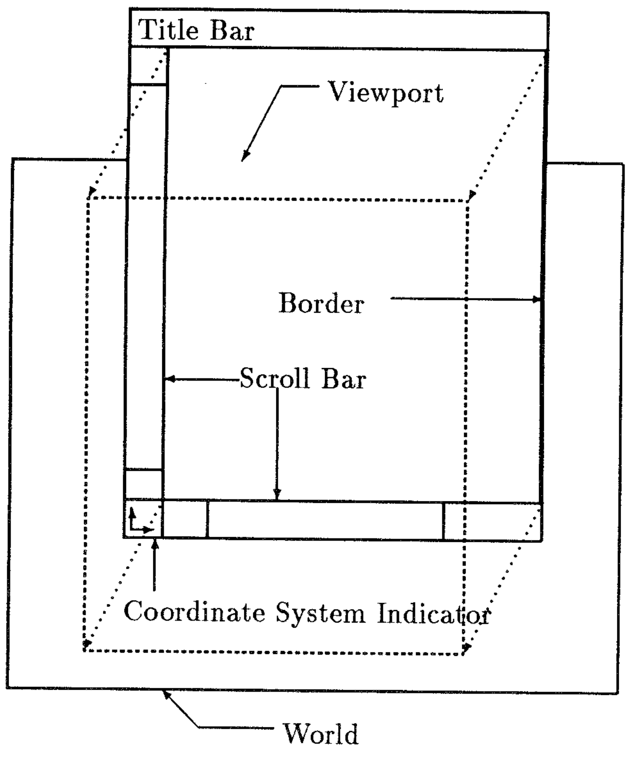


Figure 4: World, Viewport and Window

8.2 Window Creation and Drawing

Window class has a slot to indicate whether the window is visible or no. This value is determined at creation.

The size of window in a screen can be determined by user. The width and height of a frame are determined at creation. The default size of a world is initially assigned to the same value as its viewport. The default width of a border is 1 dot. The width of title bar is the same as window width. The height of title bar depends on the height of fonts for title characters.

Whether title, scroll bar and coordinate system indicator appear or no can be controlled. Most of the controlling parameters stored in the slots of window instance are dynamically changeable.

On the initial start up of *YyonX* , a root window is created and displayed. a guide window is then appeared as shown in Fig. 5. The root window and the guide window have no title bar, scroll bar, and coordinate system indicator displayed.

8.3 Icon

Icon is a display symbol which means a window is dormant. The window which is displayed as an icon is drawable. But the contents are not displayed.

8.4 Drawing to a Window

Drawing with drawing primitive is done through a graphic stream object. If the window stream class, which is a subclass of the graphic stream, is used, drawing is done to a world. If the bitmap stream class, which is another subclass of the graphics stream, is used, drawing is done to a stipple. Graphic primitives like drawing a line or a circle, and text displaying primitives are provided. Common Lisp standard input/output primitives can direct data to the window stream in *YyonX* .

Drawing to a window is to add an additional 'drawing instance' to a slot of the world instance. It means *YyWS* has no bit plane which corresponds to each window.

9 Pop up Menu Operations

Window creation invokes the set up of accessories by default. The default accessory is a pop up menu for mouse right button. With this pop up menu, window operation can be selected.

For windows other than the root, Pressing a mouse button at the title bar or border triggers to display a pop up menu. This menu offers a choice among the operations corresponding to each menu items. The pop up menu for the root window is on its frame. This menu has a same functionality as the above pop up menu, but an selecting a window operation added. The following is the list of methods on pop up menu.

Move method: This method moves the position of a window.

Reshape method: This method changes the size of a window. The size of each components of a window is changed but still kept displayed.

Expose method: This method places a window on top the window occlusion stack. The window is displayed as the top and the whole contents of it are displayed.

Bury method: This method places a window on the bottom of the window occlusion stack. The window is displayed as the bottom. The part which is placed under other windows is not visible.

Flush method: This method removes a window instance and several objects which are owned by it. These window instances are disappeared from screen.

Clear method: This method clears the contents of a window and the world which is connected to the frame.

Shrink method: This method removes a window from the screen and creates and displays an icon instead.

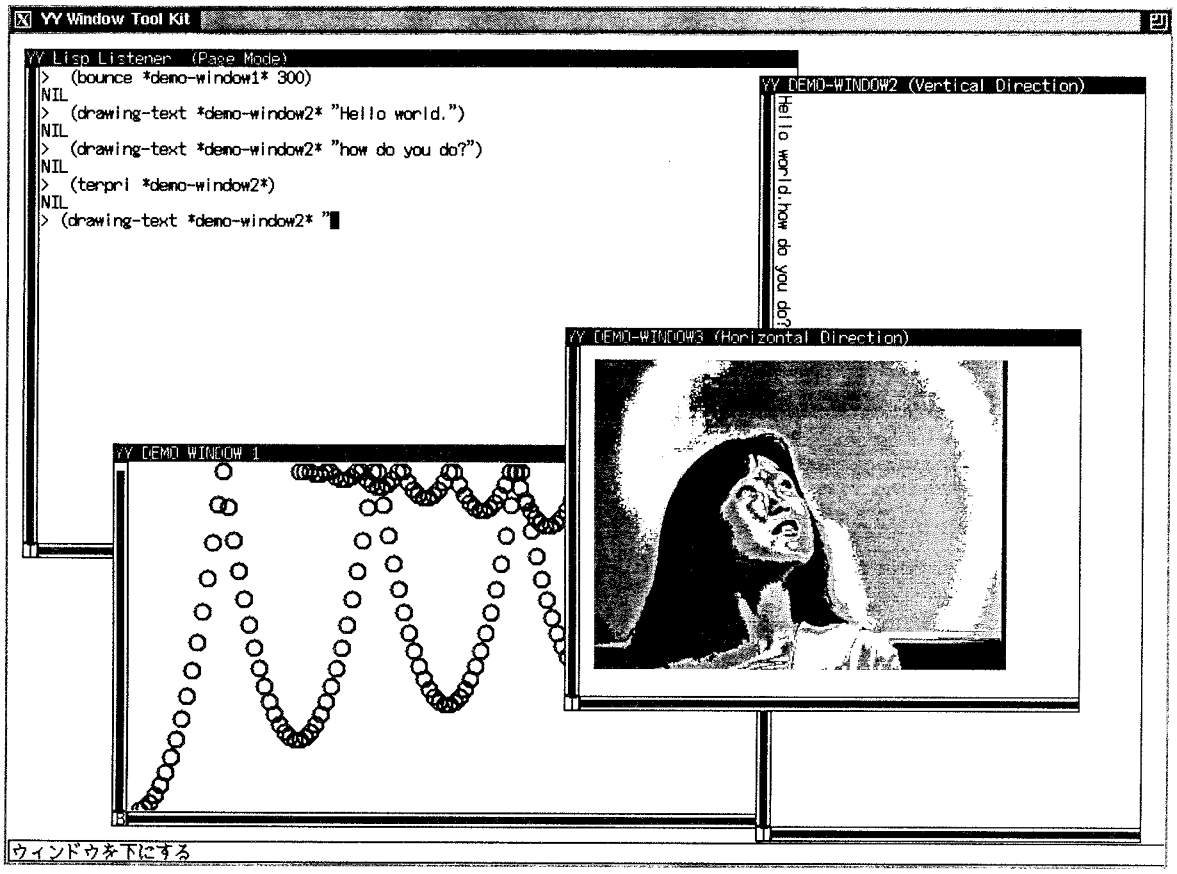
Expand method: This method removes an icon from the screen and displays the corresponding window. The window still has the same attribute as it has before. The window is displayed at the proper place, keeping its sibling relation.

10 To use *YyonX*

YyonX is started by `Initialize-yy` command. This makes a root window displayed. A root window is a window of X window. So every X window handling is applied to the root window. Fig. 5 shows an example. Multiple worlds can be handled in one root window. Multiple root window can be invoked at the same time. They are assigned different processes. The communication between them should be explicitly directed by the user.

A root window always have a guide window at the bottom. A guide window is used by input editor, system warning/guiding message, Japanese text input front-end and so on. A Lisp listener window is automatically prepared. On calling editor, GnuEmacs is invoked with the user's normal initialization.

Scroll bar and standard popup menu are provided for each window. For viewport frame, bars are appeared on the left and the bottom. For page frame, bar on the left is appeared. Standard popup menu contains primitive operations like Shrink (Expand), Move, Reshape,


YY Window Tool Kit
YY Lisp Listener (Page Mode)
YY DEMO-WINDOW2 (Vertical Direction)
YY DEMO-WINDOW3 (Horizontal Direction)
YY DEMO WINDOW 1

```
>  (bounce *demo-window1* 300)
NIL
>  (drawing-text *demo-window2* "Hello world.")
NIL
>  (drawing-text *demo-window2* "how do you do?")
NIL
>  (terpri *demo-window2*)
NIL
> (drawing-text *demo-window2* "
```

Hello world.how do you do?

ウィンドウを下にする

A root window can hold arbitrary numbers of windows inside. This example displays four windows. One Lisp listener window is automatically appeared.
A root window always have a guide window at the bottom. The guide window is used by input editor, system warning/guiding message, Japanese text input front-end and so on.
The mark appeared in the left-bottom of each subwindow must be T or B for *YyonX* version 1. T indicates the origin of the axis is the left-top, while B indicates the origin is the left-bottom.

Figure 5: An example display of *YyonX*

Clear, Expose, Bury. Furthermore, root window is given a menu of special operations to handle whole window.

11 Development Status

The development of *YyonX* started on April 1989. On October 1989, the prototype started to work.

We are evaluating the server-client model of us. With the server-client model, *YyonX* splits a Lisp application into two. It enables the reduction of CPU load and memory load of one machine. Application software can fit with smaller machines than 'all-in-one' window tool kit.

Acknowledgement

The main framework of this research was born in the discussions under the 1988 Jeida Common Lisp Committee Secretariat meetings. The authors would like to express their gratitude to all the members of JCLC.

Also thanks are due to the IFAI of Japan, ANSI X3J13 committee members, Dr. Richard Gabriel (Lucid Inc./ Stanford Univ.), Mr. Fritz Kunze (Franz Inc.), and Mr. Bill York (International Lisp Associates) for their assistances.

References

1. M.Ida: "YYAPI External Specification Manual" 1989 Dec.
2. M.Ida, T.Kosaka, K.Tanaka: "Design of *YyonX* " Proc. IPSJ annual conf., march 1990
3. M.Ida, et.al. : "A Requirement Analysis for A Portable Window System" ibid.
4. T. Kosaka, et.al. : "Design of YYWS for YYonX" ibid.
5. K. Tanaka, et.al. : "Design of YY-server for YYonX" ibid.
6. *Yy* Project : "*Yy* Protocol Specification Manual" 1989 Dec.

— No Migration but Co-existence —

Part II
Quick Introduction

1 What is *Yy*

- *Yy* is a network oriented window toolkit for Common Lisp.
- *Yy* has three layers: *YYAPI*, *YYWS*, and *NWSI*. *YYAPI* is an application program interface. *YYWS* is a window toolkit kernel. *NWSI* is a native window system interface. With the replacing *NWSI* ability, a dynamic portability of application software over the existing window systems is achieved.
- *YyonX* is an implementation of *Yy* for X-window.
- *Yy* research is carried by CSRL Aoyama Gakuin University. Prof. Masayuki Ida is the leader and the principal designer of the *Yy*.

2 What is *YyonX* ?

2.1 Overview

- *YyonX* is a pilot implementation of *Yy* for X-window, and is developed at Aoyama Gakuin Unversity.

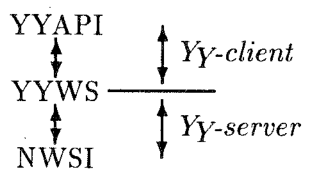
  We call it a distributed window toolkit.
- CLOS oriented portable window tool kit.
- It uses a *Server / Client* model.

  Yy-server is a X-client for low level stuffs.

  Yy-client is a X-client for an application part and API.
- There is *Yy*-Protocol for the communication between *Yy-server* and *Yy-client.*

  Yy-Protocol enables the co-operation of *Yy-server* and *Yy-client* over TCP/IP, and has a highly compressed format.
- For example, you may use a SUN3/60 X-server, a SUN4/260 *Yy-server*, and a *Yy-client* on a different machine to execute *YyonX* based application.
- Tested X-windows are X11R4, X11R3.
- Color image processing and animation features.

2.2 Server-Client Model Implementation of *Yy* Three Layers



2.3 *Yy-server* and *Yy-client*

- *Yy-server* is executable on the machines which can understand X-protocol as a X-Client, and IPC sockets. Usually *Yy-server* fit on most of UN*X machines.
  - All the Actual Images are Stored and Managed in the *Yy-server* (*Yy-client* has no bitmaps)
  - The objects of *Yy-server* are called *TERRITORY*
- *Yy-client* is executable on the machines which have a 'full' Common Lisp and have a facility for IPC sockets over TCP/IP.

  Yy-client has a sub process to dispatch events from *Yy-server*
- *Yy-server* and *Yy-client* communicate with *Yy*-protocol

2.4 *Yy* Protocol connects *Yy-client* and *Yy-server*

The characteristics of the *Yy* Protocol are:

- Currently 71 Commands
- Packet: 4 bytes × (2 + Command Arguments)
- Implemented on UNIX Socket Interface, which is facility for Inter Process Communication through TCP
- Three types of Commands:
  Notification (server → client. 3),
  Command (client → server without requiring ack. 46),
  Instruction (client → server requiring ack/answer 22)
- Transfer Rate: On SUN-4/280, 0.55msec/byte is achieved (40 byte packet transfer. avarage of 100 time experimentation).
- Intermittent Synchronization of Commands

3 Mailing List and FTP ?

There is an international mailing list for *YyonX*.
The name is `yyonx@csrl.aoyama.ac.jp`
To request, send E-mail to
`yyonx-request@csrl.aoyama.ac.jp`
YyonX sources and documents are ready for anonymous FTP at ftp.csrl.aoyama.ac.jp under *Yy* directory.

(You can make anonymous FTP to `ftp.csrl.aoyama.ac.jp` from USA, Europe and other Internet nodes.)

Sources are copyrighted but freely used acknowledging the notices on the top of each source code. Documents are placed in public domain, though we may update the contents time to time.

4 Major Components and Classes of *YY*

- Position and Region

  The logical slots for regions are :Top, :Bottom, :Left, :Right, :Width, and :Height. User can choose LBRT or LBWH as a set of physical slots.
- Presentation classes (and Output Recording)
- World, Viewport and Page
- Stream Object: Based on the amendment proposal to X3J13. `graphic-stream`, its subclass **`window-stream`** and **`bitmap-stream`** are the major streams introduced.

  `Event-stream`is provided to handle asynchronous input events.
- Active-region
- Window and Window-stream:
- Stipple: Stipple is a bitmap for bitmap-stream.
- Color-table
- Input Editor

5 Window Structure

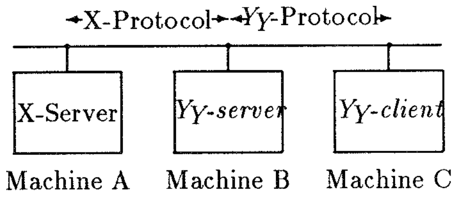
- multiple numbers of subwindows are displayed in a root window.
- multiple worlds are displayed in a root window.
- a guide window is at the bottom of a root window.
- Window consits of title and frame.
- Title can be omitted. Title and frame can have accesaries.
- Frame is viewport or page. border of a frame has width. frame supports scrolling.
- dormant window becomes an icon.
- horizontal direction and vertical direction are selectable upon string text output.

6 Three ways for window toolkit implementation

1. **as a part of Kernel:** functions are included in the OS. Ex. Lisp machines.

2. **as a library:** functions are attached to the application. Most of the window toolkits are implemented with this model. Disadvantage is on the increase of memory size and CPU loads, though heavy equipment bless users with easy to use functionalities.
3. **as a server/client cooperation:** The case for *YyonX*.
   Application program is a client of a window. Window handling is separated as a server. It enables a cooperation of different vendors' server and client.

7 Distributed Window Toolkit Concept of *Yy*



三台のワークステーション (WS) による共同動作をしめす。
応用が実行される WS、ウィンドウ上のユーザインタフェースが実行される WS、X-server が実行される WS を別にできる。

— Co-operation on Heterogeneous Environment —

X - server	*Yy* - server	*Yy* - client
UNIX	→	→
X terminal (or PC)	UNIX	Lisp machines like Symbolics
X terminal (or PC)	UNIX	Host Computers
⇑		⇑
User		Application Software
(Client)		(Lisp application server)

$$\begin{tikzcd}
\text{Other Processes} \arrow[dr, shift left] & & \text{Yy - Client} \arrow[dl, shift left] \\
& \text{Yy - Server} \arrow[ul, shift left] \arrow[ur, shift left] \arrow[d, shift left] & \\
& \text{User (X-server)} \arrow[u, shift left] &
\end{tikzcd}$$

8 Why *YY* is good: 利用者から見た利点

- Because *YY* supports the "No Migration but Co-existence" Principle.
- No need to port your applications from Symbolics to other workstations.
- No need to rewrite user interface on porting.
- No need to combine several functional modules into one executable image.
- Your Propriety is gurded while can make co-operation with other software.

9 *YY* project steps

step		
1	1989 Apr. - 1989 Sep.	Investigation & Design
2	1989 Oct. - 1990 Mar.	Prototype
3	1990 Apr. -	version 1.0 (SCL 対応)
4	1990 June	FTP announce, mailing list starts
5	1990 July	version 1.1 (Allegro 対応)
6	1990 Sept.	version 1.2 (Symbolics Genera 対応)
7	1991 Feb.	version 1.3 (imaging and animation)
...	(1991 March 予定)	version 2.0

Part III
Appendix

1 Release Note for YyonXver.1.2 dated 1990/11/05

1.1 summary

YyonXversion 1.2 is now ready for anonymous FTP at ftp.csrl.aoyama.ac.jp under Yydirectory.

1.2 files

There are several files. (Sizes for documents may change, sorry.)

```
          Nov  6 12:34 ReleaseNote.1.2
    3823 Nov  6 12:34 WHAT.IS.YYonX
  159764 Nov  6 12:55 client.1.2.tar.Z
  220155 Nov  6 12:55 client.1.2.tar.Z.uu
    6146 Nov  6 12:55 demo.1.2.tar.Z
    8501 Nov  6 12:55 demo.1.2.tar.Z.uu
  247807 Nov  6 12:55 doc.1.2.tar.Z
  341456 Nov  6 12:55 doc.1.2.tar.Z.uu
    1033 Nov  6 12:34 how-to-install-yy-client
     663 Nov  6 12:34 how-to-install-yy-server
  124737 Nov  6 12:55 server.1.2.tar.Z
  171893 Nov  6 12:55 server.1.2.tar.Z.uu
```

1. ReleaseNote.1.2 is this file. (plain text file)
2. WHAT.IS.YYonX contains the explanation of YyonX. (plain text file) Please read this file before unpack the source tar files.
3. client.1.2.tar.Z(.uu) contains the compressed tar file for yyclient directory.
4. server.1.2.tar.Z(.uu) contains the compressed tar file for yyserver directory.
5. doc.1.2.tar.Z(.uu) contains the compressed tar file for yydoc directory. There are Yy-protol documents and *YYAPI* documents. (*YYAPI* documents will be updated shortly.)
6. demo.1.2.tar.Z(.uu) contains the compressed tar file for yydemo directory. (Currently, only one file is there.)
7. how-to-install-yy-client (plain text file) contains the direction to install yy-client.
8. how-to-install-yy-server (plain text file) contains the direction to install yy-server.

Sources are copyrighted but freely used acknowledging the notices on the top of each source code. Documents are placed in public domain, though we may update the contents time to time.

1.3 Features of Version 1.2

Major features of 1.2 version are:

1. YY-client is tested with Lucid 4.0 (Lucid CLOS), Allegro 3.1.13 (PCL), Allegro 3.1.17 (International version, PCL), and Symbolics Genera 8.0 (Symbolics CLOS). YY-server is mainly tested on Sun4 SunOS4.1 and X11R4. (And is working on several other UN*X and/or X11R3 like titan.)
2. Updated YY-server to cope with new features.
3. Updated YY-Protocol (between YY-server and YY-client) specification with the introduction of the page mode (for text oriented interaction) and viewport mode (for high speed graphic outputs).
4. Event processing, popup menu and several primitives are refined.
5. Many bug fixes and improvements of 1.11alpha.

1.4 Current Plan

Further schedule of development:

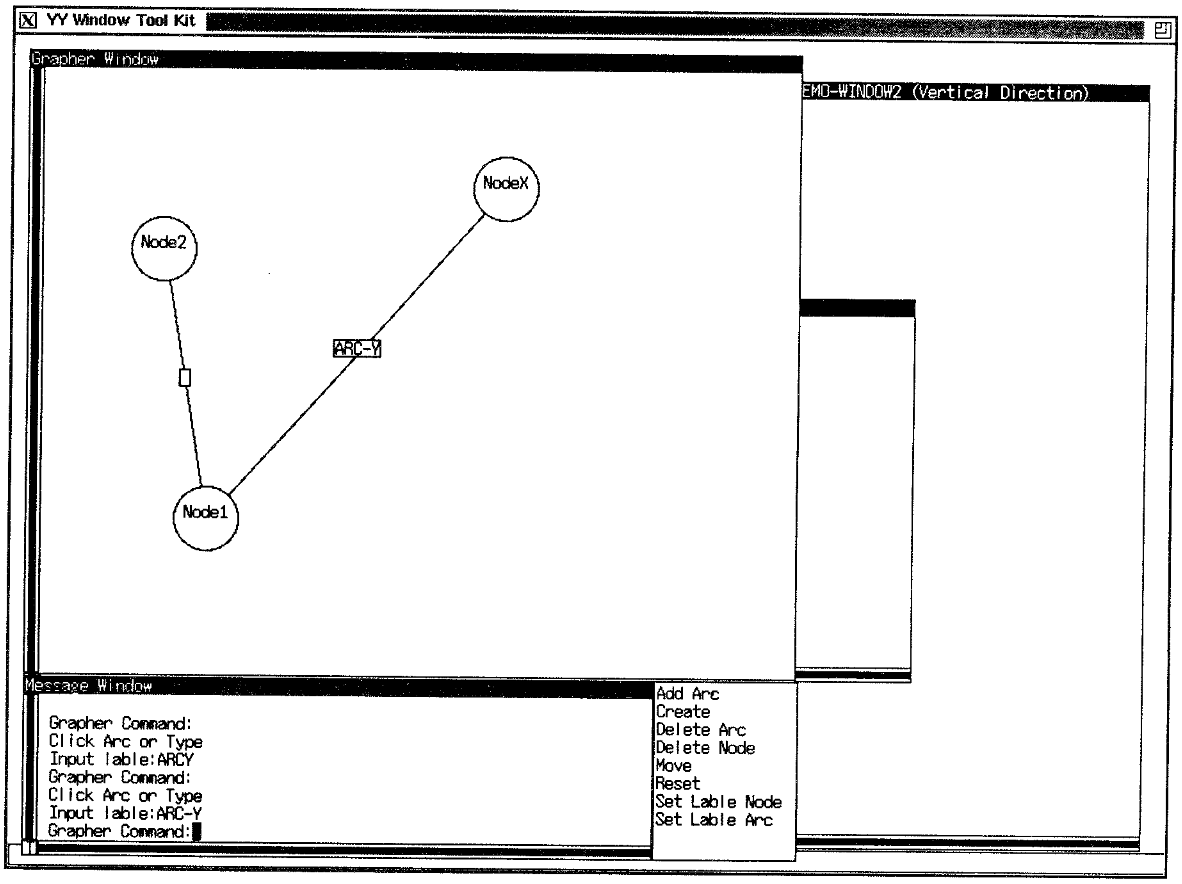
1. Refinement of *YYAPI* specification. (Yes, we had not made enthusiastic efforts on improving API specification yet, since we like to adopt standards for external interfaces and had not enough time to take care of outlooks.)
2. More enhancements on Graphics, animation area.
3. UI tools enhancements.
4. Versions for mainframe Common Lisps are under development.
5. UK researchers told us that the Eulisp is running on top of YY. We will welcome such experimentation.
6. Attempt to start the discussion with CLIM to have an unified external specification. (YYand CLIM made a joint statement for it on 1990/02/14.)
7. Continuous trial to obtain much higher performance on the communication between YY-client process and YY-server process.

2 Example: Grapher

2.1 Overview

Grapher which was originally included in the book *Lisp Lore: 2 nd edition* by Bromely and Lamson, and was adopted to the materials for comparative study of several object oriented languages making the parts of *Common Lisp Object System* by Masayuki Ida et.al.

It is an interactive software to display and manipulate the nodes and arcs.



```
Grapher Command:
Click Arc or Type
Input lable:ARCY
Grapher Command:
Click Arc or Type
Input lable:ARC-Y
Grapher Command:
```

2.2 Source Codes

```
;;;
;;;  Grapher on YYonX
;;;

;;; nodes-and-arcs.lisp

(defvar *all-the-nodes* nil)

(defvar *next-node-index* 0
  "Unique index assigned to each node.")

(defvar *next-arc-index* 0
  "Unique index assigned to each node.")

(defclass node ()
  ((arcs :initform nil :accessor node-arcs)
   (xpos :accessor node-xpos
 :initarg :xpos)
   (ypos :accessor node-ypos
 :initarg :ypos)
   (radius :initform :needs-calculation
   :accessor node-radius)
   (label :initform nil
  :accessor node-label)
   (shape :initform :circle
  :accessor node-shape)
   (unique :initform (incf *next-node-index*)
   :accessor node-unique))
  )

(defmethod initialize-instance :after ((node node) &rest initargs)
  (declare (ignore initargs))
  (push node *all-the-nodes*))

(defmethod print-object ((node node) stream)
  (let ((name (or (node-label node)
  (format nil "Unnamed node ~D" (node-unique node)))))

      (write-string name stream)))

(defmacro map-over-nodes ((node-var) &body body)
  `(dolist (,node-var *all-the-nodes*)
     ,@body))

(defmethod (setf node-label) :after (new-value (node node))
  (setf (node-radius node) :needs-calculation))

(defmethod move-node ((node node) new-xpos new-ypos)
  (with-slots (xpos ypos) node
    (setf xpos new-xpos
  ypos new-ypos)))

(defmethod delete-self ((node node))
  (dolist (arc (node-arcs node))
    (delete-self arc))
  (setf *all-the-nodes* (delete node *all-the-nodes*)))

(defmethod present-self ((node node) window)
  (calculate-radius node window)
  (with-slots (xpos ypos radius label) node
    (draw-circle-xy window xpos ypos radius)
    (draw-string-xy window (or label " ") (- xpos radius) ypos)
    ))

(defmethod calculate-radius ((node node) window)
  (with-slots (radius label) node
    (when (and (stringp label)
       (zerop (length label)))
      (setf label nil))
    (when (eq radius :needs-calculation)
      (setf radius (ceiling (string-width window (or label " ")) 2)))
    radius))


(defclass arc ()
  ((mark :initform nil
 :accessor arc-mark)
   (node1 :accessor arc-node1
  :initarg :node1)
   (node2 :accessor arc-node2
  :initarg :node2)
   (position1 :initform (make-position)
      :accessor position1)
   (position2 :initform (make-position)
      :accessor position2)
   (label :initform nil
          :accessor arc-label)
  ))

(defmethod add-arc ((node node) (arc arc))
  (incf *next-arc-index*)
  (with-slots (arcs) node
    (push arc arcs)))

(defmethod remove-arc ((node node) (arc arc))
  (with-slots (arcs) node
    (setf arcs (delete arc arcs))))

(defmethod initialize-instance :after ((arc arc) &rest initargs)
  (declare (ignore initargs))
  (with-slots (node1 node2) arc
    (add-arc node1 arc)
    (add-arc node2 arc)))

(defmethod print-object ((arc arc) stream)
  (with-slots (node1 node2) arc
      (format stream "~a <--> ~a" node1 node2)))

(defmacro map-over-arcs ((arc-var) &body body)
  (let ((mark-var (make-symbol "MARK"))
(node-var (make-symbol "NODE")))
    `(let ((,mark-var (list nil)))
       (dolist (,node-var *all-the-nodes*)
 (dolist (,arc-var (node-arcs ,node-var))
           (unless (eq (arc-mark ,arc-var) ,mark-var)
             ,@body
             (setf (arc-mark ,arc-var) ,mark-var)))))))


(defmethod delete-self ((arc arc))
  (with-slots (node1 node2) arc
    (remove-arc node1 arc)
    (remove-arc node2 arc)))

(defmethod present-self ((arc arc) window)
  (with-slots (node1 node2 position1 position2) arc
    (multiple-value-bind (x1 y1 x2 y2)
        (find-edges-of-nodes (node-radius node1)
     (node-xpos node1) (node-ypos node1)
     (node-radius node2)
     (node-xpos node2) (node-ypos node2))
      (setf (position-x position1) (min x1 x2)
    (position-x position2) (max x1 x2)
    (position-y position1) (min y1 y2)
    (position-y position2) (max y1 y2))
      (draw-line-xy window x1 y1 x2 y2))))

(defmethod present-self :after ((arc arc) window)
  (with-slots (label position1 position2) arc
    (let* ((width (string-width window (or label " ")))
   (x (- (+ (position-x position1)
                (ceiling (/ (- (position-x position2)
                               (position-x position1)) 2)))
 (ceiling (/ width 2))))
  (y (+(position-y position1)
                (ceiling (/ (- (position-y position2)
                               (position-y position1)) 2))))
  (height (font-kanji-height (stream-font window)))
  (c-base (font-kanji-base-line (stream-font window))))

      (with-graphic-state ((color graphic-color) (filled filled-type)) window
  (setf color *white-color*
filled *FillSolid*)
```

```
    (draw-region-xy window (- x 1) (- y c-base) (+ width 2) height))
      (draw-region-xy window (- x 1) (- y c-base) (+ width 2) height)
     (draw-string-xy window (or label " ") x y)
     )))

(defun find-edges-of-nodes (r1 xpos1 ypos1 r2 xpos2 ypos2)
  (let* ((dx (- xpos2 xpos1))
 (dy (- ypos2 ypos1))
 (length (isqrt (+ (* dx dx) (* dy dy)))))
    (values (+ xpos1 (ceiling (* dx r1) length))
    (+ ypos1 (ceiling (* dy r1) length))
    (- xpos2 (floor (* dx r2) length))
    (- ypos2 (floor (* dy r2) length)))))


;;;Follwings are implementation dependent functions
;;; YY implementation
;;; Coded by T.kosaka

(defun string-width (window string)
  "returns the width (pixels) of string"
  (font-string-length (stream-font window) string)
  )

;;; Window stream
(defvar *grapher-window* nil)
(defvar *command-window* nil)
(defvar *menu-window* nil)

;;; Command status
(defvar *command-status* nil)

;;; Menu List
(defvar *menu-list* '((add-arc-command "Add Arc" "Add Arc")
      (create-node-command "Create" "Create Node")
      (delete-arc-command "Delete Arc" "Delete " "Arc")
      (delete-node-command "Delete Node" "Delete " "Node")
      (move-node-command "Move" "Move")
      (reset-command "Reset" "Reset")
      (set-lable-command "Set Lable Node" "Set Lable " "Node")
      (set-lable-arc-command "Set Lable Arc" "Set Lable " "Arc")))



;;; Redisplay all nodes and arcs
(defun redisplay-all ()
  (clear-window-stream *grapher-window*)
  (map-over-nodes (node) (present-self node *grapher-window*))
  (map-over-arcs (arc) (present-self arc *grapher-window*)))

;;; selection node from mouse position
(defun select-node (x y)
  (map-over-nodes (node)
     (if (> (node-radius node)
    (sqrt (+ (expt (- x (node-xpos node)) 2)
     (expt (- y (node-ypos node)) 2))))
 (return-from select-node node))))

;;; Selection ARC from XY
(defun select-arc-xy (x y)
  (map-over-arcs (arc)
     (with-slots (position1 position2) arc
       (when (and (< (position-x position1) x)
  (> (position-x position2) x)
  (< (position-y position1) y)
  (> (position-y position2) y))
   (let ((katamuki1 (ceiling (* (/ (- (position-y position2)
      (position-y position1))
   (- (position-x position2)
      (position-x position1)))
10)))
 (katamuki2 (ceiling (* (/ (- y (position-y position1))
   (- x (position-x position1)))
10))))
   (if (> 10 (- katamuki1 katamuki2))
       (return-from  select-arc-xy arc))
   )
 ))))


;;; Selection ARC from two nodes
(defmethod select-arc ((node1 node) (node2 node))
  (map-over-arcs (arc)
   (if (or (and (eq node1 (arc-node1 arc)) (eq node2 (arc-node2 arc)))
   (and (eq node1 (arc-node2 arc)) (eq node2 (arc-node1 arc))))
       (return-from select-arc arc))))

;;; Setting command status macro
(defmacro with-command-status ((status) &body body)
  (let ((old-status (gentemp))
(ret-val (gentemp)))
  `(let ((,old-status ,*command-status*)
 (,ret-val nil))

    (with-slots (input-string) ,*command-window*
 (setf input-string nil))

     (setf  *command-status* ,status
    ,ret-val (progn ,@body)
    *command-status* ,old-status)
     ,ret-val)))

;;; Command functions
(defun create-node-command ()
  (let ((x nil) (y nil))
    (multiple-value-setq (x y)
      (catch 'position (get-xy-position x y)))
    (new-write-message (format nil "Create node ~a ~a" x y))
    (present-self (make-instance 'node :xpos x :ypos y) *grapher-window*)))

(defun get-one-node (message)
  (let ((node nil))
    (new-write-message (format nil "Click ~a node or Type " message))
    (with-command-status (:find-node)
      (setf node
  (catch 'node
    (loop
     (let ((d-node (eval (read *command-window*))))
     (if (eq (class-name (class-of d-node)) 'node)
 (return d-node))))
     )))
      node))

(defun get-one-arc ()
  (let ((arc nil))
    (new-write-message "Click Arc or Type ")
    (with-command-status (:find-arc)
      (setf arc
  (catch 'arc
    (loop
     (let ((d-arc (eval (read *command-window*))))
     (if (eq (class-name (class-of d-node)) 'arc)
 (return d-arc))))
    )))
    arc))


(defun get-xy-position (x y)
  (let ((xx nil)
(yy nil))
    (with-command-status (:position-define)
      (when (or (null x) (not (integerp x)))
             ;;; Ask x position
(select-window *command-window*)
(new-write-message "Input Node X Position or Mouse click : ")

(setf xx (read *command-window*))

      (when (or (null y) (not (integerp y)))
(new-write-message "Input Node Y Position or Mouse click : ")
(setf yy (read *command-window*))
)
      ))
    (values xx yy)
    ))
```

```
(defun add-arc-command (&rest arg)
  (let ((node1 nil)
(node2 nil))

    (unless (zerop *next-node-index*)
    (setf node1 (get-one-node "No1 ")
  node2 (get-one-node "No2 "))

    (present-self (make-instance 'arc :node1 node1 :node2 node2)
  *grapher-window*)))
  )


(defun delete-arc-command (&rest arg)
  (let ((node1 nil)
        (node2 nil))
    (unless (zerop *next-arc-index*)
    (setf  node1 (get-one-node "No1 ")
   node2 (get-one-node "No2 "))
    (delete-self (select-arc node1 node2))
    (redisplay-all))))

(defun delete-node-command (&rest arg)
  (let ((node nil))
    (setf node (get-one-node " "))
    (delete-self  node)
    (redisplay-all)))

(defun move-node-command (&optional (x nil) (y nil))
  (let ((node nil))
    (setf node (get-one-node " "))
    (setf *select-node* node)
    (enable-event *grapher-window*)

    (multiple-value-setq (x y)
      (catch 'position
(get-xy-position x y)))

    (move-node node x y)
    (disnable-event *grapher-window* *mouse-move*)
    (redisplay-all)))


(defun reset-command (&rest arg)
  (clear-window-stream *grapher-window*)
  (clear-window-stream *command-window*)
  (setf *all-the-nodes* nil
*next-node-index* 0
*nex-arc-index* 0))

(defun set-lable-command (&optional (string nil) &rest state)
  (let ((node nil))
    (unless (zerop *next-node-index*)
    (terpri *command-window*)
    (setf node (get-one-node " "))
    (new-write-message "Input lable: ")
    (setf string (read-line *command-window*))
    (setf (node-label node) string)
    (redisplay-all)))
  )

(defun set-lable-arc-command ()
  (let ((arc nil)
(string ""))
    (unless (zerop *next-arc-index*)
    (setf arc (get-one-arc))
    (new-write-message "Input lable: ")
    (select-window *command-window*)
    (setf string (read-line *command-window*))
    (setf (arc-label arc) string)
    (redisplay-all)))
  )

;;; Make Menu window
(defun make-menu-window ()
  (let* ((width (font-string-length *default-font* "Set Lable Node  "))
 (window (make-window-stream :width width
 :height (* (font-kanji-height *default-font*) 10)
 :left 500 :bottom 500
 :title-string "Menu Window"
 :title-bar-visible nil
 :vertical-scroll-visible nil
 :horizontal-scroll-visible nil
 :coordinate-area-visible nil))
 (y 0)
 (a-region nil))

   (dolist (item *menu-list*)
   (setf a-region (make-active-region :left 0
            :bottom y
     :width width
     :height (font-kanji-height *default-font*)
     :parent window))

   (setf (get (active-region-symbol a-region) 'command-name)
 (car item)
 (button1-method a-region) 'menu-command
 (get 'menu-command 'single-process) t)

   (incf y (font-kanji-height *default-font*))

   (write-string (second item) window)
   (terpri window)
```

```
    (force-output window))
    (setf *menu-window* window)))

(defmethod menu-command ((a-region active-region) state)
  (throw 'command
 (values (get (active-region-symbol a-region) 'command-name))))

;;; Make Grapher window and Command window
(defun make-grapher-window ()
  (let ((window (make-window-stream :left 10 :bottom 10
     :width 600 :height 500
     :title-string "Grapher Window"))
(c-window (make-window-stream :left 10 :bottom 500
      :width 490 :height
      (* (font-kanji-height *default-font*) 10)
      :title-string "Message Window"
      )))
    (set-window-method window
       'right-button-down
       :event-mask *mouse-right-1*)

    (set-window-method window
       'move-node-mouse
       :event-mask *mouse-move*)

    (setf (get 'right-button-down 'single-process) t
  (get 'move-node-mouse 'single-process) t
  *command-window* c-window
  *grapher-window* window)

    (disnable-event window *mouse-move*)

  ))

;;; Button method for *grapher-window*
(defmethod right-button-down ((window window-stream) state)

    (case *command-status*
      (:position-define
          ;;; Set integer to window stream read buffer
       (throw 'position (values (mouse-state-x-position state)
    (mouse-state-y-position state))))

      (:find-node
       (let ((node nil))
 (if (eq (class-name (class-of (setf node
(select-node (mouse-state-x-position state)
  (mouse-state-y-position state)))) 'node)
     (throw 'node (values node)))
 ))
      (:find-arc
       (let ((arc nil))
  (if (eq (class-name
   (class-of (setf arc
       (select-arc-xy (mouse-state-x-position state)
      (mouse-state-y-position state)))))
  'arc)
  (throw 'arc (values arc)))))

       (t
        )))

;;; move-node-method
(defmethod move-node-mouse ((window window-stream) state)
  (let ((select-node *select-node*))
    (with-slots (arcs) select-node
      (with-graphic-state ((op graphic-operation)) window
     (setf op *GXOR*)
  (present-self select-node *grapher-window*)
  (dolist (s-arc arcs)
    (present-self s-arc *grapher-window*))
  (setf
      (node-xpos select-node) (mouse-state-x-position state)
      (node-ypos select-node) (mouse-state-y-position state))

  (present-self select-node *grapher-window*)
  (dolist (s-arc arcs)
    (present-self s-arc *grapher-window*)
    )))
    ))

;;; initialize-grapher
(defun initialize-grapher ()
  ;;; Makeing grapher window
  (make-grapher-window)
  (make-menu-window)
  (setf *all-the-nodes* nil
*next-node-index* 0
*nex-arc-index* 0)
  (start-grapher))

;;; Start grapher
(defun start-grapher ()
  (select-window *command-window*)
  (let ((exec-command nil))
    (clear-window-stream *command-window*)
    (loop
     (write-message "Grapher Command: ")
     (setf exec-command
   (catch 'command
     (get-command-list)))

     (apply exec-command nil)
     (terpri *command-window*)
  )))

;;; get-command-list
(defun get-command-list ()
  (let ((item nil)
(real-menu nil)
(ret nil)
(new-menu-list *menu-list*))

    (setf item (string-capitalize (string (read *command-window*))))

    (do ((menu (car new-menu-list) (setf new-menu-list (cdr new-menu-list)
 menu (car new-menu-list))))
((null new-menu-list))
    (when (string= item (third menu) :end1 (length item)
   :end2 (length item))
  (setf real-menu menu)
  (return)))

    (clear-string-region *command-window* item)

    (write-message (nth 2 real-menu))

    (when (stringp (fourth real-menu))
  (setf item (string-capitalize (string (read *command-window*))))
  (dolist (menu new-menu-list)
  (when (string= item (fourth menu) :end1 (length item)
  :end2 (length item))
(setf real-menu menu)
                  (return))))

    (clear-string-region *command-window* item)
    (write-message (fourth real-menu))

    (values (car real-menu))))


;;; scrolling in same window
(defmethod stream-force-output :before
   ((stream window-stream))
 (when (equal stream *command-window*)
  (when (> (+ (stream-cursor-y-position stream)
      (font-kanji-height *default-font*))
   (region-height (frame-region stream)))
(clear-window-stream stream))
  ))

;;; Write Message to command window
(defun write-message (message)
    (write-string  message *command-window*)
    (force-output *command-window*))
```

```
(defun new-write-message (message)
    (terpri *command-window*)
    (write-string  message *command-window*)
    (force-output *command-window*))

;;; claear region in window stream
(defmethod clear-string-region ((stream graphic-stream) item)
  (let ((x (stream-cursor-x-position stream))
(y (- (stream-cursor-y-position stream)
      (font-kanji-base-line (stream-font stream))))
(w  (font-string-length (stream-font stream) item))
(h  (font-kanji-height (stream-font stream))))

  (with-graphic-state ((color graphic-color) (filled filled-type)) stream
     (setf color *white-color*
   filled *FillSolid*)
     (draw-region-xy stream x y w h))))
```