Design of the Nagiya Editor

Pete Ware

July 5, 1994

Contents

Chapter 1

Introduction to the Nagiya Editor

The Nagiya Editor is a graphical editor for representing an object oriented design. It allows the user to draw objects and indicate the relationship between these objects. Being a graphical editor it supports commands such as printing, moving, (re)drawing, and (re)sizing. It is implemented using Xm++, a C++ library that supports the Motif widget under the X11 Window System.

Eventually, the Nagiya Editor will allow one to specify two schemas and indicate how data stored under one representation can be translated into the other representation. The Nagiya Editor will also facilitate this translation process by providing visual feedback to the user during conflict resolution.

1.1 Acknowledgements

A number of people contributed to this effort.

Shifang Sun did much to complete this project and in particular added dialog boxes for getting file names and error reporting. He did much to make this program useful.

Bill Bradley did most of the work of reading and writing schemas as well as providing enthusiasm and a great sounding board for ideas.

Neeraj Aggarwal and Madhukar Khare developed the postscript output code.

Suzanne Wilson developed much of the code for drawing and interacting with the links.

Ram Viswananthan refined much of the code for reading and writing the schema files.

Shixun Yan and Patrick Rabau helped Shifang Sun develop the error reporting window as a C++ class.

Rajesh Kumar and Rajeev Rajan contributed to the initial design of the Nagiya editor.

Chapter 2

Manager Object Class

2.1 Purpose of the Manager Class

The Manager object class is responsible for the interactions between the Link's and the Entity's. It is the layer between the user interface and the data the user is manipulating. For example, selecting a menu item eventually causes a method (or methods) in the Manager to be executed. However, the Manager knows nothing about menus.

The Manager is not responsible for performing the actual work involved in displaying the objects.

It should be possible for multiple diagrams to be edited simultaneously. Objects could be copied from one diagram to the next using the shared selection_list.

Ideally, multiple viewpoints of the same diagram should be possible but this would require sharing the instance variable ntable between several Managers – which might require further thought.

A viewpoint is just examining the same data but from a different perspective. Example viewpoints might be a window scrolled to a different section of the diagram, different link types highlighted, or some entities not visible in a display. None of these are implemented but this is a motivation for allowing multiple Manager objects to exist.

Currently, the Manager object is created by NagiyaWindow::initialize(). The user manipulates the objects in a diagram (i.e. Link and Entity objects) typically by selecting menu items corresponding to NagiayWindow methods which then invoke approprate Manager method or methods. These NagiyaWindow methods are specific to the user interface.

2.2 Manager Class Hierarchy

The manager inherits from XmControl and XmControlPane.

```
XmObject
  XmControlPane
                    Manager
  XmManager
    XmControl
                    Manager
```

The manager is contained in the files:

include/manager.h: The header file declaring class Manager.

src/manager.C: The implementation of the Manager object.

2.3 Manager's Public Methods and Variables

Constructors

2.3.1 Manager

void

Manager::Manager(	char *	name,	)
	int	x = 0,	
	int	y = 0,	
	int	width = 300,	
	int	height = 200,	
	xmStyle	= 0	

Calls constructor *XmControl::XmControl* with arguments *(name, x, y, width, height, style)*. Creates a new `NagTable` and initializes other instance variables.

XmControlPane

2.3.2 getName

virtual const char *
Manager::getName(void)

Returns the name of this object using XmObject::getName ().

2.3.3 handle

virtual Widget
Manager::handle(void)

Returns the widget for this object. Uses XmObject::handle ().

2.3.4 asObject

virtual XmObject *
Manager::asObject(void)

Returns this Manager object cast to an XmObject.

XmObject

XmManager

XmControl

2.3.5 createControl

virtual bool
Manager::createControl(XmControlPane * parent)

Creates a xnagManager widget using parent.handle () as the parent widget. Adds Manager::handle_event() to deal with button down callback.

2.3.6 defaultCallback

virtual char *
Manager::defaultCallback(void)

2.3.7 controlType

virtual ctrlType
Manager::controlType(void)

Fix: returns ToolBar as the type.

Manager

2.3.8 save_file

int
Manager::save_file(const char* filename)

Saves the current diagram to the file `filename`.

It tells the filemgr to start using this file and then invokes Manager::write() to store the contents of the diagram.

2.3.9 retrieve_file

int
Manager::retrieve_file(const char* filename)

Adds the contents of `filename` to this diagram.

It tells the filemgr to start using this file and then invokes Manager::read() to incoporate the contents of the file.

2.3.10 print_to_file

void
Manager::print_to_file(const char * filename)

Write all the objects to a postscript file.

Creates a Printer object using `filename`. Then it invokes postscript_gen () on each of the links and entities.

2.3.11 read

int
Manager::read(void)

Reads a diagram using the filemgr.

Looks for keywords `#NAGIYAENTITY` to create a new Entity object and `#NAGIYALINK` to create a new Link object. It then uses the read() method of either object to read the rest of the object description.

2.3.12 write

int
Manager::write(void)

Writes the current diagram to filemgr.

Calls the write() methods of the Link and Entity objects to write a description of those objects. It writes a line: "#NAGIYAENTITY" for Entity objects the line: "#NAGIYALINK" for Link objects.

In particular, it writes all Entity objects first. This allows Manager::read to not have to worry about creating links to non-existant entities.

2.3.13 add

void
Manager::add(Entity * entity)

Inline method defined in `manager.h`. Adds this to the list of XmObjects managed using addObject(). Then adds it into the list of objects managed by ntable. Finally, calls the Entity::createControl() method.

See note in Link::add().

2.3.14 add

void
Manager::add(Link * link)

Inline method defined in `manager.h`. Adds this to the list of XmObjects managed using addObject(). Then adds it into the list of objects managed by ntable. Finally, calls the Link::createControl() method.

Note: The Entity and Link widgets should share a superclass. Then these methods could be combined into one.

2.3.15 handle_event

void
Manager::handle_event(void * cbinfo)

Either add a link or a point to the selection list.

If the shift key is held down, append the selected item to the list. Otherwise, clear the selection list then add the selected item to the list.

This callback is added to the Manager widget in Manager::createControl().

2.3.16 select_entity

void
Manager::select_entity(Entity_Event * event)

Add an Entity onto the selection list.

Currently not implemented.

2.3.17 select_link

void
Manager::select_link(void)

Add a link onto the selection list.

Currently not needed as this is done by Manager::handle_event.

2.3.18 select_entity_attrib

static void
Manager::select_entity_attrib(Entity_Event * event)

Add an entitye's attribute onto the selection list.

2.3.19 select_point

int
Manager::select_point(int * x_ret,)
int * y_ret

Returns the first point on the selection list, or zero if there are no points

2.3.20 select_points

```
int
Manager::select_points( int **   x_ret, )
                        int **   y_ret
```

Returns an array of points on the select list. The value returned is the number of points in the array. 0 count indicates there are no elements in the array and the array is NULL. Otherwise the caller must delete[] the allocated memory

2.3.21 select_delete_objs

```
void
Manager::select_delete_objs( void   )
```

Delete all the objects current on the selection list.

Not yet implemented.

2.3.22 clear_selection

```
void
Manager::clear_selection( void   )
```

Clears the selection list.

2.3.23 get_dim

```
void
Manager::get_dim( Dimension &   w, )
                  Dimension &   h
```

Get the current size, in pixels, of the Manager widget.

2.3.24 set_print_option

```
void
Manager::set_print_option( int   i )
```

Sets PrintOption to the value of `i`. Indicate how the diagram should be printed. A value of 0 causes portrait mode to be used. Otherwise landscape mode is used.

Portrait mode is the normal orientation of a printed page. Landscape mode is rotated so text reads along the long edge of a page.

2.3.25 get_print_option

int
Manager::get_print_option(void)

Returns the value of PrintOption.

2.4 Manager's Private Methods and Variables

2.4.1 manager

Widget manager;

The widget that is responsible for managing the display of the link and entity widgets. This is what the user actually sees on the screen.

2.4.2 filemgr

FileMgr filemgr;

Maintains the persistent storage of the diagram.

2.4.3 ntable

NagTable * ntable;

The list of Entities and Links making up this diagram.

2.4.4 selectlist

static SelectionList selectlist;

; The currently set of selected objects. It is static because the user should only be manipulating a single selection list even if there are multiple viewpoints.

2.4.5 PrintOption

int PrintOption;

If this is 0, then the diagram is printed in portrait mode. Otherwise, it is printed in landscape mode. See Manager::set_print_option for a description of landscape and portrait modes.

Chapter 3

The Entity Object Class

3.1 Purpose of the Entity Class

An entity is a box with lines of text. It is divided into two areas. The title bar and the attribute area. The title bar is just one line of text. The attribute area is a list of attributes. An attribute is a single line of text. (When fully implemented, each attribute is aware of the link('s) (if any) associated with itself. When an attribute is moved, it updates the end point position of the link.)

Information on the Entity may be saved to a named file and retrieved from the file later. The Entity may be printed.

3.2 How to create and manipulate entities

To add an entity, the user selects a point on the canvas by left-clicking the mouse and either 1) selects the Create Entity entry from the Edit menu or 2) pushes the Create Entity button on the right side of the canvas.

The pattern of the entity frame indicates the state of the entity. When the pointer is within the entity frame area, the frame changes from white to gray and the entity is ready to respond to the user. To move the entity, the user pushes the middle button, drags the entity to desired location and releases the button. To select the entity, the user clicks the left button on the frame. The entity frame will then become black indicating that the entity has been selected. The entity frame is white otherwise.

A group of entities can be selected by 1) left-clicking the frame of the first entity and then 2) shift-left-clicking the frame of the rest of the entities to be selected.

After the selection is done, the group of entities can be deleted or moved together.

3.3 Entity Class Hierarchy

The Entity inherits from XmControl.

```
XmManager
  XmObject
    XmControl
                  Entity
```

The Entity is contained in the files:

include/entity.h The header file declaring class Entity.

src/entity.C The implementation of the Entity object.

3.4 The Entity Class' Public Methods and Variables

Constructors

3.4.1 Entity

```
void
Entity::Entity( char *name,          )
                int x = 0,
                int y = 0,
                int width = 300,
                int height = 200,
                xmStyle style = 0
```

Calls constructor *XmControl::XmControl* with arguments *(name, x, y, width, height, style)*. Creates a new Entity and initializes other instance variables.

3.4.2 Entity

void
Entity::Entity(void)

Create an Entity with default arguments *("NAGIYAENTITY", 0, 0, 0, 0, 0).*

Destructors

3.4.3 ~Entity

void
Entity::~Entity(void)

Deallocates the memory allocated for the entity and destroys the widget created for the entity.

XmControl

3.4.4 createControl

virtual bool
Entity::createControl(XmControlPane *)

Create the actual user interface element (i.e., the entity).

3.4.5 defaultCallback

virtual char *
Entity::defaultCallback()

This callback returns NULL.

3.4.6 controlType

virtual ctrlType
Entity::controlType()

This callback returns type ToolBar as the control type of the object.

XmObject

3.4.7 realize

virtual bool
Entity::realize()

Draws the entity and returns TRUE.

Entity

3.4.8 is_selected

bool
Entity::is_selected()

Returns the value of the flag select_flag which indicates whether the entity has been selected or not.

3.4.9 select_set

void
Entity::select_set()

Sets the frame of the entity to a special pattern that visually marks the selection of the entity. It also sets the select_flag to TRUE.

3.4.10 select_unset

void
Entity::select_unset()

Restores the frame of the entity and resets the select_flag to FALSE.

3.4.11 set_manager

int
Entity::set_manager(Manager *manager)

Sets the entity's manager to manager. So that some functions that can't be carried out by a single entity can be passed to the manager of the entity.

3.4.12 set_frame_pattern

void
Entity::set_frame_pattern(Pixmap p)

Sets the pattern of the entity's frame to the pattern stored in pixmap p.

3.4.13 set_id

int
Entity::set_id(long ident)

Sets the id to ident. Called when a newly created entity is added into the nagiya table.

3.4.14 get_id

long
Entity::get_id()

Returns the Nagiya id number of the entity object.

3.4.15 set_title

int
Entity::set_title(char *title)

Sets the resource title of the entity's widget to the argument title. Called in createControl.

3.4.16 set_pos

int
Entity::set_pos(Position x,)
Position y

Changes the position of the entity to (x, y).

3.4.17 set_dim

```
int
Entity::set_dim( Dimension w, )
                 Dimension h
```

Changes the size of the entity to (w, h). Currently not used.

3.4.18 get_pos

```
int
Entity::get_pos( Position  x, )
                 Position  y
```

Gets the current position of the entity.

3.4.19 attribute_add

```
int
Entity::attribute_add( const char *name )
```

Adds a new attribute to the current entity. The name of the attribute is stored in a string pointed to by name argument.

3.4.20 read

```
int
Entity::read( FileMgr  filemgr )
```

Reads all the information associated with the entity from filemgr.

3.4.21 write

```
int
Entity::write( FileMgr  filemgr )
```

Writes all the information that is needed to regenerate the entity into filemgr

3.4.22 move_entity

```
static void
Entity::move_entity( Widget w,              )
                     XEvent *event,
                     String *params,
                     Cardinal *num_params
```

The entity's this point is passed to the function through the resource data of the widget w. The entity will be moved to the position whose displacement to the entity's current position is stored in the data structure pointed to by event.

3.4.23 select_entity

```
static void
Entity::select_entity( Widget w,              )
                       XEvent *event,
                       String *params,
                       Cardinal *num_params
```

Sets the curr_entity kept by the mymanager to the entity's id and passes the whole event data structure pointed to by event to the manager's handle_event to be processed.

3.4.24 enter_entity

```
static void
Entity::enter_entity( Widget w,              )
                      XEvent *event,
                      String *params,
                      Cardinal *num_params
```

Sets the frame of the entity to a special pattern that visually marks that the mouse is currently inside the entity if the entity is not currently selected; otherwise, does nothing.

3.4.25 leave_entity

```
static void
Entity::leave_entity( Widget w,              )
                      XEvent *event,
                      String *params,
                      Cardinal *num_params
```

Resets the frame of the entity if the entity is not currently selected.

3.4.26 done

```
static void
Entity::done( Widget w,              )
              XEvent *event,
              String *params,
              Cardinal *num_params
```

Flush all changes to the entity on the screen to visually realize them.

3.4.27 set_sel_attrib_str

```
void
Entity::set_sel_attrib_str( void *cbinfo )
```

Sets member selected_attrib_str to the name selected attribute. The usage of this method is not clear at this moment because the association between entity and link has not been implemented.

3.4.28 attrib_sel_call_back

```
void
Entity::attrib_sel_call_back( void *cbinfo )
```

Put the attribute selected into the selection list. Attributes on the selection list are not used at this moment for the same reason as the previous method.

3.4.29 postscript_gen

```
void
Entity::postscript_gen( Printer *p )
```

Draws the entity on p. Called by the entity's manager.

3.4.30 single_postscript_gen

```
void
Entity::single_postscript_gen( Printer *p )
```

Draws the entity on p and realizes it. Called by entity itself.

3.5 Entity Class' Private Methods and Variables

3.5.1 mymanager

Manager * mymanager;

The window manager which controls the entity.

3.5.2 selected_attrib_str

char * selected_attrib_str;

The name of the selected attribute.

3.5.3 id

long id;

The id number of the entity.

3.5.4 etitle_string

char * etitle_string;

The name of the entity.

3.5.5 e_x

Position e_x;

The x coordinate of the entity.

3.5.6 e_y

Position e_y;

The y coordinate of the entity.

3.5.7 e_w

Dimension e_w;

The width of the entity.

3.5.8 e_h

Dimension e_h;

The height of the entity.

3.5.9 attr_name[MAX_ATTRIB]

char * attr_name[MAX_ATTRIB];

The list of the entity's attribute names.

3.5.10 etitle

Widget etitle;

The entity's title bar widget.

3.5.11 row_col_list[MAXATTRIB]

Widget row_col_list[MAXATTRIB];

The array of RowColumn widgets. Each element contains the attribute name (a Text widget) and a pushbutton.

3.5.12 text[MAX_ATTRIB]

Widget text[MAX_ATTRIB];

The array of Text widgets. Each element displays the name of an attribute.

3.5.13 buttons[MAX_ATTRIB]

Widget buttons[MAX_ATTRIB];

The array of PushButton widgets. Each element is a push button related to one of the attributes.

3.5.14 workwin

Widget workwin;

The working window in which the list of attributes are put.

3.5.15 num_attributes

int num_attributes;

The number of attributes.

3.5.16 read_op

bool read_op;

If it is TRUE, this entity is being read in from a file.

3.5.17 del_flag

bool del_flag;

If it is TRUE, it means the del button is pushed.

3.5.18 select_flag

bool select_flag;

If it is TRUE, this entity has been selected by the manager.

3.5.19 scroller

Widget scroller;

The Scrolled window which contains the list of attributes.

3.5.20 vertical

Widget vertical;

The vertical scrollbar alongside the list of the attributes.

3.5.21 scroll_value

int scroll_value;

The scrollbar's slider position.

3.5.22 scroll_slider_size

int scroll_slider_size;

The scrollbar's slider size.

3.5.23 scroll_increment

int scroll_increment;

The scrollbar's slider increment.

3.5.24 scroll_page_increment

int scroll_page_increment;

The scrollbar's slider page increment.

3.5.25 AddDelCB

void
Entity::AddDelCB(void *data)

This function takes care of the work to be done if the user clicks on either the add or delete button. When the add button is pushed, a new attribute entry is added to the end of the attribute list.

When the delete button is pushed, the entity informs the associated link. (currently not implemented).

When the print button is pushed, this entity is printed.

3.5.26 ScrollCB

void
Entity::ScrollCB(void *data)

Callback function when the scrollbar is being used. Currently not fully implemented.

Chapter 4

OWindowBuf Object Class

4.1 Purpose of the OWindowBuf Class

The OWindowBuf object class is a subclass of the class streambuf (see man page *sbuf.pub*(3) and *sbuf.prot*(3)). OWindowBuf class is a public interface of character buffering class, it supports buffers into which characters can be inserted. When the content of the buffer needs to be consumed, the OWindowBuf outputs it to a window.

4.2 OWindowBuf Class Hierarchy

The owindowbuf inherits from streambuf.

The owindowbuf is contained in the files:

include/owindowbuf.h: The header file declaring class OWindowBuf.

src/owindowbuf.C: The implementation of the OWindowBuf object.

4.3 OWindowBuf's Public Methods and Variables

Constructors

4.3.1 OWindowBuf

```
void
OWindowBuf::OWindowBuf( Widget tw,                      )
                        Widget ow,
                        int flag = TRUE,
                        int bsize = OWINBUFMAX
```

Initialize instance variables text_w, owindow, cursor_position, unitbufset. Initialize buffer and its put area.

Destructors

4.3.2 ~OWindowBuf

```
void
OWindowBuf::~OWindowBuf( void )
```

Inline method defined in OWindowBuf.h. Deletes buffer.

streambuf

4.3.3 overflow

```
virtual int
OWindowBuf::overflow( int c )
```

Overwrites the virtual function declared in class streambuf. Inserts content of the put area into the text widget text_w. Shows the display window if it is hidden. Initializes buffer and its put area.

4.3.4 sync

```
virtual int
OWindowBuf::sync( void )
```

Overwrites the virtual function declared in class streambuf. If unitbufset is set (meaning buffer is flushed after each insertion), does nothing. Otherwise, calls *OWindowBuf::overflow* to consume content of buffer. In Nagiye Editor, since owindowstream always sets the flag unitbufset, so sync is called only at the end of the program (when the series of destructors are called).

4.4 OWindowBuf's Private Methods and Variables

4.4.1 owindow

Widget owindow;

The parent of text widget text_w.

4.4.2 text_w

Widget text_w;

The text widget that displays content of the buffer so far.

4.4.3 cursor_position

XmTextPosition cursor_position;

The current cursor position of text_w.

4.4.4 buffer

char * buffer;

The buffer where data is put into.

4.4.5 bufsize

int bufsize;

The size of the buffer.

4.4.6 unitbufset

int unitbufset;

When this is true, it means that the stream associated with this buffer has the flag unitbuf set to true.

Chapter 5

OWindowStream Object Class

5.1 Purpose of the OWindowStream Class

The OWindowStream class supports insertion (storing) into an OWindowBuf. These are referred to as output operations. Everything going into this OWindowStream object goes into buf and is consumed later (output to the display window).

The display window of an OWindowStream object has a text widget with a scroll bar and an OK button. Whenever something is output to the stream, the display window will appear with all data output so far. When the OK button is pushed, the display will disappear. Later on, when more data is output to the stream, the window will reappear.

Since OWindowStream class is a subclass of ostream_withassign, which is in turn a subclass of ostream, so there is no need to redefine the operator << and any other output operations. Therefore, OWindowStream also accepts any user-defined data types.

The reason OWindowStream is derived from ostream_withassign instead of ostream is that the the class ostream (of *AT&T* library) does not have a constructor which takes no arguments, while ostream_withassign has one which does no initialization and this allows consturction of class objects to be done later.

5.2 OWindowStream Class Hierarchy

The owindowstream inherits from ostream_withassign.

```
ios
  ostream
    ostream_withassign
                      OWindowStream
```

owindowstream is contained in the files:

include/owindowstream.h: The header file declaring class OWindowStream.

src/owindowstream.C: The implementation of the OWindowStream object.

5.3 OWindowStream's Public Methods and Variables

Constructors

5.3.1 OWindowStream

void
OWindowStream::OWindowStream(void)

Inlined method defined in OWindowStream.h. Does nothing here. Members are to be set later.

5.3.2 OWindowStream

void
OWindowStream::OWindowStream(char * name,
Widget parent)

Calls *OWindowStream::initWin* to initialize class instance variables.

OWindowStream

5.3.3 initWin

```
void
OWindowStream::initWin( char * name,     )
                        Widget parent
```

Calls *OWindowStream::setupWidgets* to create a text widget text_w. Initializes the buffer and link it with the superclass ios (which bp is a member of). Sets the unitbuf flag so that the buffer is flushed after each insertion.

5.3.4 hideTextCB

```
static void
OWindowStream::hideTextCB( Widget,                )
                           OWindowStream * obj,
                           XtPointer
```

Inline method defined in OWindowStream.h. Hides the display window. Note: Non-static member function of C++ is always passed the *this* pointer as an implicit argument, this creates a problem when one wants to use the member function as a callback function of some widget. One of the methods to get around the problem is to declare the member function to be static (like we did here) to avoid the *this* pointer being passed.

5.4 OWindowStream's Private Methods and Variables

5.4.1 setupWidgets

```
void
OWindowStream::setupWidgets( char * name,          )
                             Widget shell_parent,
                             Widget &text_w
```

Sets up the layout of the display window.

5.4.2 getTopShell

```
static Widget
OWindowStream::getTopShell( Widget w  )
```

A convenient tool to find the top shell of widget w. (not a private method, where should it go?)

5.4.3 buf

OWindowBuf * buf;

A pointer to the buffer - an instance of OWindowBuf.

5.4.4 owindow

Widget owindow;

The display window, a shell widget of type topLevelShellWidget.

Chapter 6

The Printer Object Class

6.1 Purpose of the Print Class

The Printer class provides the methods for creating a hardcopy of the contents of a Nagiya screen. There are primitive functions here provided to the manager class which enable the following to be achieved.

- Set up the page output such that output can be produced in one of three modes, portrait, landscape or multipage.
- Print to a file. The manager class contains a public method called print_to_file which uses the methods in the Printer class to produce an output.

6.2 A Brief Description of the Implementation Details of the Various Print Modes

There is an integer variable called PrintOption which allows the manager to decide which option is to be used for printing. If PrintOption is 0 then the Printer is configured in landscape mode. If it is 1 then the Printer is configured in portrait mode. Otherwise it is configured in the multipage output mode.

There exist specific functions in the Printer class which can configure the Printer in one of these options. The following design issues had to be tackled. ... complications,

- The coordinate system used by the Nagiya Editor is different from that used by Postscript. In Postscript we are restricting our attention to the first quadrant of a coordinate system, However in any windowing environment, including the Nagiya Editor, the coordinate system restricts ones attention to the fourth quadrant of the coordinate system.
- As a consequence of the above it is necessary to shift the coordinate axes by a certain amount for the portrait mode. In the landscape mode it is necessary to shift as well as rotate the coordinate system so as to produce a proper output. For generating output according to the multipage mode, one must produce a number of coordinate shifts and produce an output in each case.
- For the portrait and the landscape modes, it is necessary to fit the entire Nagiya Window contents into one sheet of paper, It is necessary to perform an appropriate scaling before the output is produced.

The methods of this class are very specific to the Postscript language. It is not very easily extensible for other (non Postscript) output types.

6.2.1 Multipage output: How it works

The multipage output mode simply works because of the following concept. When the coordinate axes are situated at the origin, then whatever lies within a distance of PAGE_HEIGHT northwards and a distance of PAGE_WIDTH eastwards will appear on the printed page. Any object at a distance greater than these quantities will not appear on the printed page.

However when one shifts ones coordinate system to a different point, then *from that point* whatever lies within a distance of PAGE_HEIGHT in the north direction and within a distance of PAGE_WIDTH in the east direction will be printed on the page.

So the idea is to print a page, shift the coordinate system so that the next page is aligned to be printed, and print that page. Repeat this process till all pages are printed.

The bulk of the code that does this shifting process exists in the public method of the manager class Void Manager:: print_to_file.

6.3 How the Print class works

As has been noted before, the Nagiya Editor consists of entities and links. Each entity may have several attributes. The Print class provides the primitive functions to create a representation of the Nagiya Editor on plain paper. It does this by translating print commands from the entity and link into postscript and creates a postscript file that draws the calling object(i.e, either entity or link).

6.4 The Printer Class' Public Methods and Variables

Constructors

6.4.1 Printer

void
Printer::Printer(char *fname)

This constructor sets width and height according to the dimensions of the page selected. These dimensions cannot be changed. They are the constants INCH_WIDTH and INCH_HEIGHT, which are initialized in the file ../include/print.h It also opens the file fname.

Destructors

6.4.2 ˜Printer

void
Printer::˜Printer(void)

Closes the open file corresponding to the Printer class. Sets file_open to 0. The variable file_open is used by the method open_file of the Printer class to determine if an file is open or not before attempting to open a file.

Printer

6.4.3 open_file

```
void
Printer::open_file( char *name )
```

The above method opens the file given by name and returns an error into nagerr if there is an error. If there is no error, a postscript file having the name name is created and appropriate headers/definitions are written. Sets file_open=1.

6.4.4 close_file

```
void
Printer::close_file( void )
```

Called only by the destructor of this class. Closes the file corresponding to the Printer class.

6.4.5 write

```
void
Printer::write( char *buffer )
```

Writes the contents of a buffer to a file. Currently not used.

6.4.6 write_line

```
int
Printer::write_line( int x1, )
                     int y1,
                     int x2,
                     int y2
```

Creates the postscript code corresponding to a line drawn from the point `(x1,y1)` to the point `(x2,y2)`. Returns 1 if successful. The width of the line is a default quantity that is beyond the control of the user.

6.4.7 write_str

int
Printer::write_str(int x1,)
int y1,
char *str

Writes the postscript code corresponding to writing the text string `str` at the point `(x1, y1)`.

6.4.8 set_xyscale

void
Printer::set_xyscale(float xscale1,)
float yscale1

Used in portrait and landscape modes. Scales the Nagiya Editor window contents such that all the entities and links fit into one page.

6.4.9 translate

void
Printer::translate(int x,)
int y

Translates a distance (x,y) relative to the current position. This translation is given in units of $\frac{1}{72}$ of an inch.

6.4.10 set_grayscale

void
Printer::set_grayscale(float grayscale1)

Sets the grayscale of the printed output to grayscale1. It is currently not used anywhere. The default value of the grayscale has been set equal to 1.0.

6.4.11 set_linewidth

void
Printer::set_linewidth(float linewidth1)

Sets linewidth on the printed output to linewidth1. This is currently set equal to the default value of 1.

6.4.12 rotate

void
Printer::rotate(int angle)

Used in landscape mode. Rotates the coordinate system by angle degrees. The landscape mode required the entire Nagiya Window to be rotated 90 degrees, this function accomplishes this task.

6.4.13 set_font

void
Printer::set_font(char *name_font1,)
int scale_font1

Sets the font to name. Not used currently. The font used currently as the default is the Palatino-Roman font.

6.4.14 realize

void
Printer::realize(void)

Draws a page and goes onto the next page.

6.4.15 set_pagedim

void
Printer::set_pagedim(float newwidth,)
float newheight

Allows you to reconfigure the page size.

6.4.16 surround_box

void
Printer::surround_box(void)

Draws an outer box for portrait and landscape modes. (Not called by multipage output mode.)

6.4.17 portrait_setup

```
void
Printer::portrait_setup( Dimension w, )
                         Dimension h
```

Sets up the page so that the output is in portrait mode. This involves shrinking the page such that one of the dimensions fits either the width or the height of the page. The aspect ratio is preserved in this operation.

6.4.18 landscape_setup

```
void
Printer::landscape_setup( Dimension w, Dimension h )
```

Sets up the page so that the output is in landscape mode. One of the dimensions exactly fits the height or the width of the page.

6.4.19 multipage_setup

```
void
Printer::multipage_setup( Dimension w,                )
                          Dimension h,
                          int          no_rows
                          int          no_columns
```

Sets up postscript so that output is produced on many pages if the dimensions are greater than PAGE_HEIGHT and PAGE_WIDTH.

6.4.20 next_page

```
void
Printer::next_page( int rows,    )
                    int columns
```

Writes a complete page (if invoked after one page has been composed) and translates the coordinate system such that subsequent pages can be printed.

6.4.21 reloc

int
Printer::reloc(int y)

This relocates the y coordinate. The Nagiya Window and the Postscript page use different coordinate systems. The postscript coordinate system has the origin at the bottom left corner of the page and the coordinates increase along the north and east directions.

The Nagiya Window coordinate system has the origin at the topleft corner of the screen. The coordinates increase along the *south* and *west* directions.

Reloc controls the mapping betweeen the two coordinate systems.

6.5 Printer Class' Private Methods and Variables

6.5.1 file_open

int file_open;

Keeps track of whether we have opened the file.

6.5.2 file

ofstream file;

The output file(with the .ps extension).

6.5.3 grayscale

float grayscale;

The grayscale in which the drawing will be done.

6.5.4 xscale

float xscale;

The current scaling factor of the x direction.

6.5.5 yscale

float yscale;

The current scaling factor in the y direction.

6.5.6 xtrans

int xtrans;

This is the current x translation.

6.5.7 ytrans

int ytrans;

This is the current y translation.

6.5.8 linewidth

float linewidth;

The current linewidth.

6.5.9 name_font[50]

char name_font[50];

The name of the font in which the text will be written.

6.5.10 scale_font

int scale_font;

Essentially the size of the font.

6.5.11 width

float width;

Width of the page.

6.5.12 height

float height;

Height of the page.

6.5.13 PrintOption

int PrintOption;

This is set equal to

- 0 for landscape mode.
- 1 for portrait mode.
- 2. for multipage output mode.

6.5.14 filename

char * filename;

The postscript file.

Chapter 7

PrintDialog Object Class

The PrintDialog class supports the simple function of creating a panel of radio buttons so that the user can set up the page output in any required format.

7.1 The PrintDialog Class Hierarchy

The PrintDialog class is a derived class of the Xm++ class XmUserDialog. It supports the following methods in addition to the methods of the XmUserDialog.

The definition of the PrintDialog class is contained in the files

include/print.h : The header file declaring the class PrintDialog

src/print.C : The implementation of the PrintDialog object.

7.2 PrintDialog Class' Public Methods and Variables

Constructors

7.2.1 PrintDialog

```
void
PrintDialog::PrintDialog( XmObject *parent, )
                          void *m
```

Calls XmUserDialog with parameters as "Print Options" and parent. It sets the private member variable manager to m.

7.3 Print Dialog Class' Private Methods and Variables

7.3.1 initWindowSize

```
boolean
PrintDialog::initWindowSize( int  x, )
                             int  y,
                             int  w,
                             int  h
```

Currently sets window size to 400×80 returns TRUE.

7.3.2 createContents

```
void
PrintDialog::createContents( void )
```

Adds new radio buttons in the window just created. These radio buttons are

Landscape If the user wants to set up according to Landscape mode.

Portrait If the user wants to set up according to the Portrait mode.

Multipage If the user desires to have multipage output.

Done When the operation of selecting the mode for page output is over.

7.3.3 initContents

void
PrintDialog::initContents(void)

A function that sets the states of the radio buttons just created as in the above to TRUE provided it senses a call back.

7.3.4 rbChanged

void
PrintDialog::rbChanged(char *name)

Sets the PrintOption of the manager to 0, 1 or 2 depending on which radio button was pressed in the above. For description of the variable PrintOption see the previous chapter.

7.3.5 closeIt

void
PrintDialog::closeIt(void)

Deletes the PrintDialog Window. This is one of the operations performed on receiving a call back from the Done radio button.

Chapter 8

The Link Object Class

8.1 Purpose of the Link Object Class

A link is a visual connection between entities. The Link object class is responsible for maintaining information to allow the user to create, alter, and destroy links. When this class is fully implemented, the user will be able to change the shape of links. Links may also be saved, retrieved, and printed.

The user creates a link by clicking with the left mouse button for the first point and shift-clicking with the left mouse button for all succeeding points. After choosing the points, the user must choose Create Link from the Edit menu. A link will then appear. The link has its own identifier; connections are established internally by forming relationships among links and entities. The Link object class has many methods for relating entities by using links. Typically a link is drawn from one entity to another entity. Each of these entities has specific associated methods.

Information on the Link may be saved to a named file and retrieved from the file later. The Link may be printed.

The Link object class is not fully implemented. Several routines have been implemented and tested, but not integrated into the overall program. It is not yet possible to make certain changes to the link (e.g., bend the link), even though the low-level code has been implemented.

Since the Link is a visual connection, Link.c graphics-oriented subroutines written in C are required (see Chapter ***).

8.2 Link Object Class Hierarchy

The Link Object Class inherits from XmObject.

```
XmObject
  XmManager
    XmControl
      Link
```

The Link is contained in the files:

include/link.h: The header file declaring class Link.

src/link.C: The implementation of the Link object.

8.3 Link Object's Public Methods and Variables

Constructors

8.3.1 Link

void
Link::Link(char *name = "link",)
int x = 0,
int y = 0,
int width = 300,
int height = 200,
xmStyle = 0

Calls constructor XmObject with arguments *(name, x, y, width, height, style)*. Creates a new `Link` and initializes other instance variables.

XmObject

8.3.2 realize

bool
Link::realize(void)

This method is required by XmObject; it normally creates a window. For a link, there is no window; one cannot realize a gadget.

XmControl

8.3.3 createControl

bool
Link::createControl(XmControlPane *par)

Creates the actual user interface element (i.e., the link).

8.3.4 defaultCallback

char *
Link::defaultCallback(void)

This callback returns NULL. The "usual" callback.

8.3.5 controlType

ctrlType
Link::controlType(void)

Ignore this routine. It is wrong.

Link

8.3.6 set_id

int
Link::set_id(long ident)

Sets the identifier for the link. Returns the value of 1.

8.3.7 get_id

long
Link::get_id(void)

Returns the identifier for the link.

8.3.8 set_to_obj

int
Link::set_to_obj(long id)

Sets an identifier for the entity (to_obj) to which one end of the link is attached. Returns the value of 1.

8.3.9 get_to_obj

long
Link::get_to_obj(void)

Returns the identifier of the entity (to_obj) to which one end of the link is attached.

8.3.10 set_to_attrib

int
Link::set_to_attrib(const char* attrib)

Sets a string descriptor for an attribute in an entity (to_obj). Returns the value of 1.

8.3.11 get_to_attrib

const char*
Link::get_to_attrib(void)

Returns a string descriptor for an attribute in an entity (to_obj) as set by set_to_attrib.

8.3.12 set_from_obj

int
Link::set_from_obj(long id)

Sets an identifier for the entity (from_obj) to which one end of the link is attached. Returns the value of 1.

8.3.13 get_from_obj

long
Link::get_from_obj(void)

Returns the identifier of the entity (from_obj) to which one end of the link is attached.

8.3.14 set_from_attrib

int
Link::set_from_attrib(const char* attrib)

Sets a string descriptor for an attribute in an entity (from_obj). Returns the value of 1.

8.3.15 get_from_attrib

const char*
Link::get_from_attrib(void)

Returns a string descriptor for an attribute in an entity (from_obj) as set by set_from_attrib.

8.3.16 intersect

long
Link::intersect(Position x, Position y)

The x and y coordinates are the mouse click position. If intersection is detected, the identifier of the entity from which the link connects is returned. If intersection is not detected, the value returned is 0.

8.3.17 bend_add

void
Link::bend_add(int where,)
Position x,
Position y

Adds a bend in the link. Takes the index to set the position; the first position in the array points is 0. Note: This routine also can be used to change a position.

8.3.18 bend_delete

void
Link::bend_delete(int where)

Removes a bend from the link. Passes in the index of the bend to delete; the first position in the array points is 0.

8.3.19 read

int
Link::read(FileMgr& filemgr)

Reads link information from a file. For format, see write below.

8.3.20 write

int
Link::write(FileMgr& filemgr)

Writes link information to a file. The information written is in the following format:

```
id
to_attrib, to_obj
from_attrib, from_obj
points[0].x  points[0].y
points[1].x  points[1].y
     ...
points[n].x  points[n].y
$$
$$

In the file {\sf #NAGIYALINK} will precede the information written, e.g.,

#NAGIYALINK
762749842
(null),0
```

```
(null),0
267 67
360 59
$$
$$
```

8.3.21 postscript_gen

void
Link::postscript_gen(Printer *p)

Generates information for PostScript.

8.4 Link Object's Private Methods and Variables

8.4.1 bend_update

void
Link Object::bend_update(void)

Changes values in the array of points representing the lines of the link.

8.4.2 points

XPointArray points;

The type XPointArray contains a list of points. These points represent the end points of the lines forming the link. Each point other than the first and the last may be regarded as a bend in the link. Points may be added or deleted from this list using bend_add or bend_delete.

8.4.3 num_points

int num_points;

The number of points needed to represent the lines of the link. This number changes as points are inserted or deleted using bend_add or bend_delete.

8.4.4 id

long id;

The identifier for the link.

8.4.5 to_obj

long to_obj;

The identifier of the entity this is linked to.

8.4.6 from_obj

long from_obj;

The identifier of the entity this is linked from.

8.4.7 to_attrib

char * to_attrib;

An attribute of the entity this is linked to.

8.4.8 from_attrib

char * from_attrib;

An attribute of the entity this is linked from.

Chapter 9

The Link Widget Class

9.1 Purpose of the Link Widget Class

The Link widget class is responsible for drawing the lines representing links on the screen. It also calculates information for changes in the location and attributes of these links. When this class is fully implemented, the user will be able to change the visual appearance of links.

The user creates a link by clicking with the left mouse button for the first point and shift-clicking with the left mouse button for all succeeding points. After choosing the points, the user must choose Create Link from the Edit menu. A link will then appear. The environment establishes this visual connection using a gadget, a windowless widget that draws in its parent's window.

These routines involve the display the of the link, the sizing of the window, and click detection. The graphics context may be recalculated; an XOR drawing function has been used to allow fast redisplay when a line is changed. In order to test when a user has clicked on (or sufficiently near) a line, a bounding box algorithm is projected. Information about the link is stored in the Link widget. Additional information is in the Link object (see Chapter ****).

The Link widget class is not fully implemented. Several routines (or parts of routines) have been tested, but not integrated into the overall program. It is not yet possible to make certain changes to the link (e.g., change the line width or the line style) or to use the bounding box for click detection, even though the low-level code has been implemented.

9.2 Link Widget Hierarchy

The Link Widget inherits from RectObj.

```
Object
  RectObj
    NagLink
```

The widget class Link is contained in the files:

include/X11/Xnag/LinkP.h: A header file containing class record and instance record for widget class Link.

include/X11/Xnag/LinkRec.h: A header file containing resources and definitions for widget class Link.

src/Link.c: Routines for widget class Link implemented in C.

9.3 Link Widget's Public Methods and Variables

RectObj

9.3.1 Initialize

```
static void
Link::Initialize( Widget request,      )
                  Widget new,
                  ArgList arglist,
                  Cardinal *numargs
```

Initializes an instance of this object.

9.3.2 SetValues

```
static Boolean
Link::SetValues( Widget current,      )
                 Widget request,
                 Widget new,
                 ArgList args,
                 Cardinal *numargs
```

Controls reset of values for foreground, line width, and line style. Calls routines to get new graphics contexts. Displays lines using XOR graphics context. NOTE: Changes for foreground, line width, and line style are not yet integrated into program.

9.3.3 Redisplay

```
static void
Link::Redisplay( Widget gw,      )
                 XEvent *event,
                 Region region
```

Redraws lines using default drawing function.

NagLink

9.3.4 XnagLinkIntersection

```
int
Link::XnagLinkIntersection( Widget gw,      )
                            int x,
                            int y,
                            XPoint *bend_point
```

A bounding box algorithm is used to detect intersection. If the user clicks within this box, the routine returns the value of 1, otherwise the routine returns the value of 0. The user is not permitted to click at the end of a link. If the click is on the link, the x and y coordinates of the nearest link end-point are stored in bend_point. The bend_point otherwise will contain (-1, -1). NOTE: This routine has not been integrated into the program.

9.4 Link Widget's Private Methods and Variables

9.4.1 CalcGC

```
static void
Link::CalcGC( Widget gw, )
              GC *w_Xor
```

Calculates graphics contexts. The permanent context is stored in the widget gw. The XOR context is returned in *w_Xor. Frees and allocates graphics contexts. Calculates new contexts based on the foreground and background attributes (or any other appropriate ones).

9.4.2 GetNewSize

```
static void
Link::GetNewSize( Widget gw )
```

Calculates new X, Y, Width and Height based on the array of points representing the link and asks parent for a new size. It doesn't matter what the parent says.

9.4.3 fg

Pixel fg;

The foreground color.

9.4.4 points

XPointArray points;

The list of points in the link.

9.4.5 count

int count;

The number of points in the link.

9.4.6 ls

int ls;

The line style for the link.

9.4.7 lw

int lw;

The line width for the link.

9.4.8 gc

int gc;

The graphics context for drawing the lines of the link.

9.4.9 pressbutton

XtCallbackList pressbutton;

The callback list for button press.