The following people and organizations have kindly granted their permission for the use of figures from their publications.

Figures 2.2.1.1 and 2.2.1.2 are from [7]. ©1981 ACM. They are used with the permission of the Association for Computing Machinery and A. Borning.

Figure 2.2.1.3 is from [8]. ©1986 Lawrence Erlbaum Associates, Inc. It is used with the permission of Lawrence Erlbaum Associates, Inc. and A. Borning.

Figure 2.2.2.1 is from [17].

Figure 2.2.3.1 is from [14]. ©1984 IEEE. It is used with the permission of IEEE.

Figure 2.2.4.1 is from [9]. ©1989 IEEE. It is used with the permission of IEEE.

Figures 2.2.5.1, 2.2.5.2, and 2.2.5.3 are from [26]. They are used with the permission of T.D. Kimura.

Figures 2.2.6.1 and 2.2.6.2 are from [40]. ©1986 McGraw-Hill, Inc., New York, NY. All rights reserved. They are used with the permission of Byte Magazine.

Figure 2.2.7.1 is from [2]. ©1987 IEEE. It is used with the permission of IEEE.

Figures 2.2.8.1 and 2.2.8.2 are from [29]. ©1988 IEEE. They are used with the permission of IEEE.

Figures 2.2.9.1 and 2.2.9.2 are from [33]. They are used with the permission of J. Rasure.

Figures 2.2.10.1 and 2.2.10.2 are from [23]. ©1992 IEEE. They are used with the permission of IEEE.
Appendix A -- Pen Interface

Getting the WACOM HD 648 pen tablet interfaced with the SPARCstation IPX was one of the significant and time consuming aspects of the implementation. References to the actual hardware used are in the main text of the thesis. This appendix is intended to serve as the "Implementation Notes" for the pen interface. It is a loosely related set of notes intended to aid in the understanding and future extension of the system.

A.1 Hardware Description and Setup

<table>
<thead>
<tr>
<th>Tablet</th>
<th>WACOM HD-648A LCD Integrated Tablet, WACOM Co., Ltd.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Card</td>
<td>Vigra VS10 Color Frame Buffer for SPARCstation Computers, Vigra, Inc.</td>
</tr>
</tbody>
</table>

The WACOM has a number of parameters to be set to allow it to communicate effectively with various computer systems. The operation handbook describes how to set the various parameters. The following parameters were used for our implementation.

TABLET

Data Format: ASCII (ASC)
Operating Mode: STREAM
Coordinate Type: ABSOLUTE (ABS)
Always Transmit: DISABLED (DSB)

Baud Rate: 19200
Parity: NONE
Stop Bits: 1
Data Bits: 8
Terminator Type: Carriage Return Only (CR)

Report Rate: Maximum (MAX)
mm/inch: millimeters (mm)
Buzzer: OFF
Origin: UPPER LEFT

DSR: Disabled (DSB)
VIDEO

RGB Signal: Red Only (R)
Display Mode: Positive (POS)
Model: PS/2

Other settings (BACK LIGHT and MAINTENANCE) should not effect the program. A MACRO setting named SUN has been created which contains most of these settings. However, the Report Rate setting seems to revert back to 100.

The output from the tablet is connected to the SPARCstation through a serial port, in this case ttya. The WACOM digitizing tablet includes a VGA compatible display. Output to the display goes through the Vigra video card.

Sometimes the video card needs to be reset. This is different than setting the parameters noted above. One piece of evidence that this needs to be done is when nothing shows up on screen when the program is run and all the above parameters are set correctly.

To reset the Vigra board, login to an account, but do not let OpenWindows start. At the command prompt enter the following:

% openwin -dev /dev/vs10fb0

This will start OpenWindows using the Vigra VGA board as the display. Exit from OpenWindows as you normally would. The Vigra board should be properly reset.

A.2 tablet.c

A very small piece of code was written in C to be used to access stylus/pen position information from the WACOM tablet. The code is named tablet.c. This code consists of the following 10 functions:
flush_input - clears the input buffer of the tablet
initWACOM_ - initializes access to the WACOM tablet
readWACOM_ - reads one packet of information from the WACOM tablet and stores the results locally so they can be retrieved by the following functions
get_WACOMdev - returns WACOM device information; type of stylus
get_WACOMx - returns most recently retrieved x coordinate; in millimeters
get_WACOMy - returns most recently retrieved y coordinate; in millimeters
get_WACOMsx - returns most recently retrieved x coordinate converted into “screen” coordinates, that is in screen pixels
get_WACOMsy - returns most recently retrieved y coordinate converted into “screen” coordinates, that is in screen pixels
get_WACOMbsp - returns most recently retrieved Button/Switch/Pressure data

There are two modes for retrieving input from a serial port (terminal port) in a UNIX environment, canonical and noncanonical. There are also two different modes for the WACOM tablet to send data, binary and ASCII. The Canonical-Binary combination was tried unsuccessfully. The Noncanonical-Binary combination was also tried unsuccessfully. It was initially assumed that ASCII transmission of data would prove to be too slow. However, both the forms of binary transmission resulted in lost bytes of data and therefore incorrect information. The Canonical-ASCII combination proved to work and be fast enough that the user doesn’t notice any delay between writing and seeing the results on screen. The book *Advanced Programming in the UNIX Environment* by W. Richard Stevens (Addison-Wesley, 1992) proved to be very helpful in finally getting this interface to work.

To be used in the SPARC-Oberon environment the code in tablet.c must be compiled into a shared library. The following commands are used.
% cc -O -c -pic tablet.c
% ld -o tablet.so tablet.o

The shared library file tablet.so is created and can be opened for use by Oberon code.

A.3 vigra-driver.c

Another small piece of code was written in C to allow access to the Vigra video card and thus to the tablet’s display. The code, in a file called vigra-driver.c, contains the following functions:

- **vigra_init** - initializes the interface to the Vigra board; opens access to the framebuffer and retrieves necessary framebuffer information
- **vigra_dot** - displays a dot on the screen; either white (for erasing) or black for displaying; inputs are color and x and y coordinates
- **vigra_line** - uses vigra_dot to display a line on the screen; Bresenham’s algorithm is used to determine the dots in the line; inputs are color and two sets of x and y coordinates
- **vigra_dashedline** - displays a dashed line; inputs same as above
- **vigra_rectangle** - displays a rectangle on the screen; inputs are color, upper left hand corner coordinates and width and height
- **vigra_dashedrectangle** - displays a dashed rectangle; inputs same as above
- **vigra_clear** - clears the display
- **vigra_dump** - uses TIFF code available in the public domain (ftp’d from telva.ccu.uniovi.es = 156.35.31.31) to create a TIFF file containing a description of the current screen

To be used in the SPARC-Oberon environment the code in vigra-driver.c must be compiled into a shared library. The following commands are used.

% cc -O -c -pic vigra-driver.c
% ld -o vigra-driver.so vigra-driver.o /usr1/tlb1/tiff/libtiff/libtiff.a

The shared library file vigra-driver.so is created and can be opened for use by Oberon code.
Appendix B -- Oberon Implementation Notes

This appendix describes the Oberon implementation more extensively than was done in the previous sections. The type hierarchy for the implementation is described in the main text of the thesis. This Appendix contains a short description of each module in the system.

Although the Oberon language contains no reference to the concept of a class, a defined type with an associated set of type-bound procedures as allowed by the Oberon-2 Language is a construction that roughly corresponds to the class concept of C++. Any references to "class" or "classes" in the following are intended to be read as a type with its type bound procedures. Similarly, references to subclassing are intended to be read as subtypes of such types.

B.1 Debug

The Debug module contains a small set of procedures which can be used to display debugging output only when the On variable is true. This allows one to insert useful debugging statements without having to go back later to delete or comment out the statements.

B.2 GraphicUtils

The GraphicUtils module contains a few procedures used by the drawing procedures for the box and arrow types. These procedures come from Computer Graphics Software Construction: Using the Pascal Language by John R. Rankin (Prentice Hall, 1989).
B.3 Vigra

The Vigra Module contains all the procedures used to display graphics using the Vigra VS10 Video board. This module contains procedures which directly correspond to those in the vigra-driver library of C code. Before using any of these procedures, the Init procedure must be called. Init uses the Kernel.dlopen procedure of SPARC-Oberon to open the shared library and uses the Kernel.disym to dynamically link Oberon symbols to symbols in the library. Init is called automatically whenever this procedure is loaded. This module also contains a few other graphics related procedures.

B.4 MyFonts

The MyFonts module contains procedures for loading a font from a standard bdf (binary distribution format) file. No font caching is implemented in this procedure, i.e. only one font, the active font, can be loaded at a time. Memory allocation for fonts is also not optimized in this module. The most significant procedures for a user of this module are SetFont, which reads the font file and establishes a valid active font, DrawCharacter, which draws a character of the active font on screen, DrawString, which draws a string of characters, and DrawInt, which converts an integer to a string and then draws the string.

B.5 WACOMInput

The WACOMInput module is used to retrieve input about the pen/stylus status from the WACOM tablet. It opens the shared library created by the tablet.c file. As in the case of the Vigra module, WACOMInput uses the Kernel.dlopen procedure of SPARC-Oberon
to open the shared library and uses the Kernel.dlsym to dynamically link Oberon symbols to symbols in the library. This module is automatically initialized when it is loaded. The most used procedure in the module is Pen. Pen checks the current status of the pen on the tablet and returns an indication of whether the other information returned is valid or not. The information will be invalid if the pen is not in proximity to the tablet. Pen coordinates and button status are returned along with the time of reading these values.

B.6 CMath

The CMath module makes some mathematical functions which are not a part of Oberon available by accessing a shared library of code written in C which implements these functions. The C program file is called CMath.c and is compiled into a shared library and opened for use by Oberon in a manner similar to the vigra-driver.c and tablet.c codes. The procedures available via CMath include SQRT, EXP, LOG, SIN, COS, and TAN.

B.7 Origin

The Origin module maintains the current absolute screen coordinates of the origin point of the diagram. This was added to allow for the future extension of the program to include scrolling through a diagram.

B.8 Objects

The Objects module creates the base type of the type hierarchy. The Object type created is what would be called by C++ programmers an abstract class. It should be subtyped (subclassed) to be used.
B.9 SortedLists

SortedLists creates an abstract list type. The types available include a List type and a Node type which implements the important functions of a list node. As is indicated by the name, the listed items are maintained in a sorted order. The order in which they are maintained is based on the implementation of the LT and EQ procedures which are used to compare list nodes. Subtypes (subclasses) of the Node type should override the define LT and EQ procedures to determine the ordering of nodes in the list.

B.10 Trees

Trees creates an abstract tree type. The types available include a Tree type and a Node type which implements the important functions of a tree node. The children of any given node are maintained in a sorted list. Thus the Trees module is heavily dependent on the SortedLists module. When new nodes are inserted into a tree, the procedure ShouldParent is used to determine whether the new node should be a child of a given node. This procedure should be overridden to determine the parent-child relationship in any implement Tree type.

B.11 LocatedObjects

This module creates the class LocatedObject. This is a subclass of the Object class. It is the superclass of all objects which have some logical location in the boxgraph diagram whether the objects are visible or not.
B.12 LocatedValues

A LocatedValue is a LocatedObject which has a single value. LocatedValues are not displayed. The LocatedValues are used for binding of parameters and passing outputs back as inputs in the implementation of the Iteration Box.

B.13 Shapes

The Shapes module creates the abstract class which is the superclass of all shapes in a diagram. Procedures for drawing, erasing, marking, unmarking, selecting, deselecting, copying, and resizing of shapes are type bound to the Shape class. Thus all subclasses must implement these operations.

B.14 Lines

The Lines module creates the class Line which contains implementations of the above Shape procedures for a Line. Also included as part of the Line class are procedures for checking the distance from a point to a line and determining whether two lines intersect.

B.15 Arrows

The Arrow class created by this module is a subclass of Lines. Procedures for drawing an arrow's head are included in this module.
B.16 Rectangles

The Rectangles module creates the class Rectangle which is a subclass of Shape. Procedures for drawing, erasing, marking, etc. rectangles are type bound to the Rectangle type. Also included are procedures for checking the distance from a point to a rectangle and determining the number of times a specified line intersects a rectangle.

B.17 HFBboxes

This module creates the class HFBBox which is the simple box type used in the Simple Hyperflow language. This class is a subclass of the Rectangle class. Procedures for setting and retrieving a box’s value, transparency (open or closed), and consistency status are bound to the HFBBox type. Also included are procedures for associating arrows with a box.

B.18 HFPredBoxes

This module creates a subclass of the HFBBox class. The HFPredBox class is the predecessor box type.

B.19 HFSuccBoxes

This module creates a subclass of the HFBBox class. The HFSuccBox class is the successor box type.
B.20 Options

This module creates the class Option which is a LocatedObject which can be drawn or erased and has an option name and an action to be carried out when the option is selected. The name and action are procedure types which are part of the record which defines the Option type. This is the Oberon (non-Oberon-2) method of associating procedures with a type. This allows the action and name procedures associated with an Option to be changed at run-time. These Options are used to implement the menu of options available to the user. Being able to change an Option’s name and action is very useful in this context.

B.21 LocatedValueLists

This module subclasses the SortedList class to form a list of LocatedValues. Lists of this type are used extensively in the implementation of the Iteration box.

B.22 ArrowLists

This module subclasses the SortedList class to form a list of Arrows. All arrows in a boxgraph program are maintained in a list of arrows. Also, the various box types form lists of their associated arrows when setting up for the execution of a program.

B.23 OptionLists

This module subclasses the SortedList class to form a list of Options. The OptionLists implements the list of options given to the user. In order to do so, the module
contains a procedure, called WhichOption, which, when passed the coordinates of a point, returns the Option to which those coordinates belong. Calling WhichOption and then calling the "action" procedure of the returned Option performs the action that the user has selected.

B.24 HFBoxTrees

This module subclasses the Tree class and the Trees.Node class to form a tree of HFBoxes. All boxes in the system are maintained in such a tree. The parent-child relationship corresponds to box containment. The ShouldParent type bound procedure is overridden to establish this relationship. Procedures are also included in this module for performing such things as determining in which box a given screen point resides, selecting and deselecting entire subtrees, determining the box in the tree which is closest to a given screen point, determining which boxes in the tree are intersected by a given arrow, and performing whatever processing is necessary on arrows which cross through box boundaries.

Also included in this module are the procedures which are used when executing a boxgraph program. The ParseTree procedure takes the current list of arrows in the diagram and makes all necessary associations between boxes and arrows. The SetConstants procedure determines which subtrees of the BoxTree represent natural number constants input by the programmer. After parsing the tree, the ExecuteNode procedure is used to perform program execution. The ReduceTree procedure uses the SetConstants and ExecuteNode procedures to execute the entire program represented by the Tree.
B.25 HFIterBoxes

This module subclasses both the HFBox class and the HFBoxTree class to create the Iteration box. The HFIterBox emulates the dynamic expansion of an iterated set of boxes by first breaking arrows which cross the boundary of the iteration box into two arrows one of which arrives at the box boundary and the other of which leaves from the box boundary. Then input values are retrieved and the contents of the iteration box are executed. Output values are retrieved and passed back as inputs. The entire subtree contained in the iteration box is reset and executed again. This process continues until the iteration box becomes inconsistent. Then the output values are made available for processing.

B.26 ShapeRecognizer

This module is the Oberon implementation of the shape recognition algorithm described in “Recognizing Multistroke Geometric Shapes: An Experimental Evaluation” by Apte, Vo, and Kimura [5]. The only procedure of this module which is used by other modules is the Recognize procedure. Recognize takes an array of point coordinates and returns an indication of the recognized shape and the parameters of the shape (origin, height, and width). This version of the algorithm recognizes lines, rectangles, circles, ellipses, and diamonds. Triangle recognition is not implemented.
B.27 BGE

The BGE module is the main module of the application. It maintains a tree of boxes, a sorted list of arrows, and a sorted list of user options. The command Init is invoked by the user to begin the application. After setting up the screen, the pen input, and the fonts, the Init procedure calls the procedure which takes over for the main event processing loop of the Oberon system. This procedure is called PenLoop. PenLoop checks for pen events and dispatches these events for appropriate handling.

PenLoop uses the WACOMInput module to check the pen status and then determines whether the user is drawing or selecting an option. If the user is drawing, then pen input is traced on the display. After the pen is lifted from the surface of the tablet, the set of input points is passed to the shape recognition algorithm. If the shape is recognized as a line, a corresponding arrow is added to the list of arrows. If the shape is recognized as any other shape, a corresponding box is added to the tree of boxes. If the user has selected an option, that option is determined and its action is executed.

Since PenLoop takes over for the main Oberon system loop, it must perform the garbage collection that is normally performed by the main loop. This is done by calls to the Kernel.GC procedure at regular intervals.
Appendix C -- User’s Guide

This appendix serves as a very brief user’s guide to the current implementation. It assumes a familiarity with the SPARC-Oberon system. After entering the directory in which the source code resides, user’s should use the following command to start SPARC-Oberon.

```
% oberon -h 22
```

This makes the Oberon heap size larger than the default. This is useful for loading fonts.

Begin interaction with the application by invoking the BGE.Init command. Once you have invoked BGE.Init, you can only interact with the system via the WACOM tablet. However, you should be careful as keystrokes and mouse button presses are saved in an input buffer which is acted upon when the BGE.Init command is finished.

To draw boxes and arrows to create a program, simply use the pen/stylus to draw the shapes on the tablet. Lift the pen from the tablet upon finishing the drawing of a shape. All shapes recognized as anything other than a line will be interpreted as a box. You can take advantage of this by indicating lines by simply tapping at the end points and indicating boxes by simply tapping at three of the four corners of the box. The other options presented in the option list can be selected by tapping in an option box. The options available are described in the following paragraphs.

The Exit option does what one would expect; it terminates the session. The Dump option creates a TIFF file containing a representation of the current contents of the screen. The Save option creates a text file containing a description of the current set of boxes and arrows. The Read option reads such a text file into the current set of boxes and arrows. The name of the file read and saved is BG.File. By accessing another command window (other than the window in which SPARC-Oberon is running), the user can rename files that are created or read in this manner. The Clear option empties the current set of boxes and
arrows. The Reduce option runs the boxgraph program created by the current set of boxes and arrows. The Solid/Dashed option indicates the current state of the Solid/Dashed flag. When this option is named Solid, the boxes drawn on screen will be solid (i.e. closed.) When this option is named Dashed, the boxes drawn on screen will be dashed (i.e. open.) Selecting this option toggles the state of the Solid/Dashed flag. The BoxType option effects the state of the BoxType flag. This flag determines what type of box is created when the shape recognizer indicates that a box has been drawn. The option and the flag have four possible states: BoxNorm (a standard HFBBox is created), BoxPred (a predecessor box is created), BoxSucc (a successor box is created), and BoxIter (an iteration box is created).

The Select option changes the overall state of the application from drawing mode (in which the user creates shapes on screen) to selection mode (in which the user can select groups of on screen shapes for deleting). When the Select option is selected, it is replaced by the Deselect, Delete, and Draw options. While in selection mode, the user selects shapes for manipulation by tapping the pen inside or near the shape. Selected shapes are marked on screen. Tapping the pen near an already selected shape, removes the shape from the set of selected shapes. Choosing the Deselect option empties the set of selected shapes. Choosing the Delete option removes all selected shapes from the diagram. The Draw option changes the application back to drawing mode.
Timothy B. Brown

VITA

Date of Birth: 7/7/65

Place of Birth: Indianapolis, Indiana

Undergraduate Study: Department of Mathematics
Butler University, Indianapolis, Indiana
Bachelor of Science, 1987

Graduate Study: Department of Computer Science
Washington University
St. Louis, Missouri, 1987-Present
Master of Science expected December, 1993.

Honors/Awards: Elected to membership in Kappa Mu Epsilon
(Mathematics Honor Society)
Elected to membership in the Honor Society of Phi Kappa Phi

Professional Experience: Student Assistant/Technical Computer Assistant
International Ground Water Modeling Center
Holcomb Research Institute of Butler University

Computer Systems Manager/Programmer
Behavior Research Lab
Department of Psychology
Washington University
1987-Present

December, 1993