The Brown Workstation Environment: A User Interface Design Toolkit

Steven P. Reiss
John T. Stasko

Department of Computer Science
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-89-34
June 21, 1989

The Brown Workstation Environment: A User Interface Design Toolkit

Steven P. Reiss
John T. Stasko

Department of Computer Science
Brown University
Providence, RI 02912-1910
(401) 863-7600

June 21, 1989

Abstract

The Brown Workstation Environment is a graphics toolkit that currently runs on top of the X Window System. It provides an extensive set of graphics management programming tools for simple input and output, menus, windows, text editing, terminal emulation, and structured diagrams. Its implementation stresses ease of use by the programmer by providing higher-level, simplified packages as well as the usual low-level components.

1 Introduction

In order to make effective use of the graphical capabilities of workstations, programmers must have an effective set of tools that make creating a user interface simple and quick. The growing popularity and availability of workstations has led to a relatively large number of different toolkits that attempt to provide varying degrees of functionality to the programmer. We at Brown have evolved a comprehensive toolkit over the past six years. This toolkit differs from most of those that have been developed in terms of the functionality it provides and its emphasis on being as simple to use as possible. We also wish to emphasize the fact that our toolkit has been in use for the past six years, both inside and outside Brown. This differs significantly from many user interface design systems that have never moved beyond the prototype stage and were used to develop interfaces for relatively few applications.

Our set of tools, collectively called the Brown Workstation Environment or BWE, was developed to support a wide range of applications, from student projects to large software systems such as PECAN[15], BALSA[5], THINKPAD[22], GARDEN[16], FIELD[17], and TANGO[25]. Its development and evolution paralleled these projects so that the functionality it provided was determined by the actual needs of the projects. BWE is especially useful for graphics intensive applications such as visualization tools and graphics editors that go beyond simple interaction methods such as windows and menus. The toolkit's development occurred on a variety of workstations including Apollos, Suns, and Microvaxes, under a variety of windowing systems. In the last year, we have migrated the environment to utilize the X11 window system[23] as a basis for portability.

BWE is strictly an interface toolkit, as opposed to a user interface management system or UIMS. In general, user interface management systems tend to present a limited number of interface styles, generally for less development effort than with toolkits. For more information on distinguishing the two, see [13]. The fact that UIMS's seldom receive wide use and acceptance has recently led to questions of their value[21, 10]. As expected, we belong to the toolkit "camp"; the flexibility evident in toolkits coincides to our varying application needs.

Our goals throughout the development of BWE were twofold. The first was to provide the functionality necessary for quality user interfaces. We needed to support a variety of menuing techniques including pop-up menus, pull-down menus, static menus and scroll bars. We wanted to provide text input and display facilities as well as program-controllable text editing. We needed to handle window layout and placement within an application. We wanted to make providing help information simple in order to encourage applications to do so. We needed the interface to work correctly in an environment that supports lightweight processes. We also needed some unusual functionality for particular applications such as the ability to rapidly display structured pictures describing data structures or programs.

The second goal in the development was to make using BWE as simple as pos-

sible for the programmer. Other toolkits such as the X11 kit and library[11] often encounter criticism for the difficulty involved in developing relatively straightforward user interfaces, especially for novice designers. BWE was developed to provide a much simpler paradigm for creating quality interfaces, and we do believe that BWE presents an attractive alternative for interface designers not wishing to probe the depths of the X11 toolkit.

Making BWE simple to use directed our development in two ways. The first was to provide fairly high-level packages rather than low-level components. While low-level components provide a greater degree of flexibility to the programmer, we felt that we could provide enough flexibility in our packages and, at the same time, greatly simplify their use. By breaking down the toolkit into packages for specific functions such as input, graphics, menus, and so forth, we partition the conceptual design model into manageable chunks. A designer is not confronted with hundreds of similar appearing entry points.

The second way the simplification goal affected BWE was in the type of interfaces it produced. We started with procedural interfaces to all the packages. These tended to be clumsy and somewhat difficult to use. The current version of BWE has evolved these interfaces be much more data-oriented. That is, menus, dialog boxes, window geometries, editor characteristics, and so on, are all defined by user data structures rather than by a set of procedure calls.

2 Overview of BWE

The Brown Workstation Environment is written as a set of packages that are called by the application program. These packages are typically interrelated, using other parts of the environment as appropriate. The packages range from a low-level graphics interface to a complete text editor. A summary of the functionality provided by the packages is shown in Figure 1. The individual packages are:

- **ASH** is the basic graphics package provided by BWE. It provides a single bitmap abstraction for both on-screen and off-screen windows. It offers a full range of drawing primitives and attributes in the windows. ASH also supports basic cursor operations, region highlighting based on mouse position, 2-D animation primitives, and full hierarchical window manipulations. All of this is implemented so that it can be used safely with lightweight processes.

- **BIO** is the basic input processing package. It takes the stream of input events provided by ASH and provides mapping, masking, and queuing facilities. It will run correctly with lightweight processes, distinguishing the different threads of control by the windows they own and providing separate queues for each thread.

- **RIP** further refines input handling in BWE by partitioning input into regions within ASH windows. Input is received on a per region basis by registering

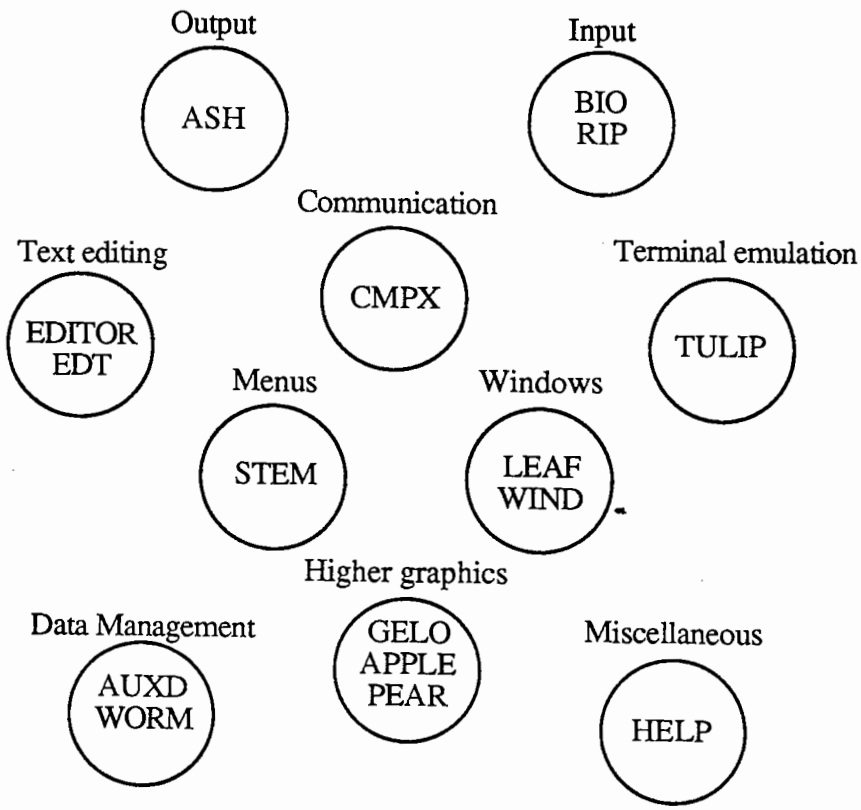


Figure 1: Overview of the functionality provided by BWE along with the component packages that implement individual capabilities.

an input handling routine and an input mask with each region. The routine is called whenever an event that is accepted by the mask occurs in the region. Programs typically use the package to provide an inverted flow-of-control where the input handler automatically calls the program only when input occurs. This facility emphasizes modular programming since any routine is apt to be called at any time. RIP is widely used by the rest of BWE.

- **CMPX** is the main selection mechanism of BWE. It allows a routine to be associated with a file descriptor or a time interval. When there is activity on the file or the time interval passes, the registered routine will be called. CMPX provides an entry point that is the principal idle loop for a BWE application.

- **STEM** provides menu capabilities for application programs in BWE. It creates an environment including sticky, Macintosh-style pull-down, pop-up, and state menus, along with dialog boxes and scroll bars. STEM generally uses a call-back approach for menus. The application associates a routine with a menu button. When the user selects that button, the routine is called with appropriate parameters.

- **LEAF** provides facilities for layout and maintenance of ASH windows. It allows the programmer to define the window layout by means of a static table that provides layout constraints and information. LEAF then does the window drawing along with any refreshing and resizing that become necessary.

- **WIND** provides basic window management facilities within BWE. Unlike the X11 window managers, it is designed to allow an in-program windowing system. That is, it is designed to provide windowing facilities for use within an application that requires multiple windows. WIND includes the capabilities to offer Macintosh-style buttons around the effective window, to do nested window management, and to create a control panel showing window options and icons as appropriate for an application.

- **EDITOR** provides a complete back end for text editing. It contains two major modules, FILE and TEXT, which perform file manipulations and screen handling, respectively. The package provides the hooks for a classic screen editor utilizing an infinite quarter-plane edit window, plus full mouse and menu capabilities. EDITOR was designed to be very flexible so that various front-ends such as syntax directed and language editors could be built on top of it.

- **EDT** provides comprehensive and extensible mouse-oriented editor capabilities within BWE. It functions as a front-end to the EDITOR package. Given an ASH window, EDT opens an edit pad in the window. Various types of pads, such as file edit windows, blank form windows, shell windows, and uncataloged pads are available. EDT offers a command definition language that

allows new commands to be defined through external, dynamically loaded C programs.

- **TULIP** is the virtual terminal emulation component of BWE. It provides full ANSI compatibility and multiple, variable-spaced fonts within an ASH window. TULIP is primarily used by the EDITOR component of BWE.
- **GELO, APPLE**, and **PEAR** provide capabilities for the drawing and display of pictures of program data structures. The display is done automatically using topological constraints. GELO actually does the drawing with a minimum of effort by the application. APPLE is a front-end that allows the user to interactively define how to draw a type for GELO. PEAR allows user editing of GELO pictures by offering hooks to the underlying application so that editing the pictures actually changes the underlying application.
- **HELP** provides a set of routines that offer a consistent interface to displaying help information to the user in BWE. It supports a hierarchy of help information that the user can peruse. HELP also processes dynamic requests for help from BWE components and application programs.
- **AUXD** is a simple database package designed to provide an application with a flexible means of accessing constant data without having to bind the data in. It can also function as a general means of storing and hierarchically accessing data.
- **WORM** is a simple object-oriented package that sits on top of C. Unlike C++, it does not include (yet) a preprocessor. It does, however, support dynamic definition of types, multiple inheritance, I/O, and dialog-style I/O.

As stated earlier, the components of BWE are very interrelated. Many are written largely using other BWE packages. In Figure 2 we present a diagram sketching the interconnection between modules. High-level components are placed at the top; lines connecting two components indicate that the upper package utilizes routines from the lower package.

In the next section we look at some of the more interesting and unique aspects of the Brown Workstation Environment by studying some of these packages in more detail. A complete description of all the packages is found in the BWE manual[6]. In Section 4 we review some related user interface toolkits that are available. Finally, Section 5 describes the current status and future directions of this work.

3 BWE Components

3.1 RIP – Region Based Input

RIP is the primary input package used by application programs in BWE. RIP's programmer interface is different than many other systems in that it is region-based

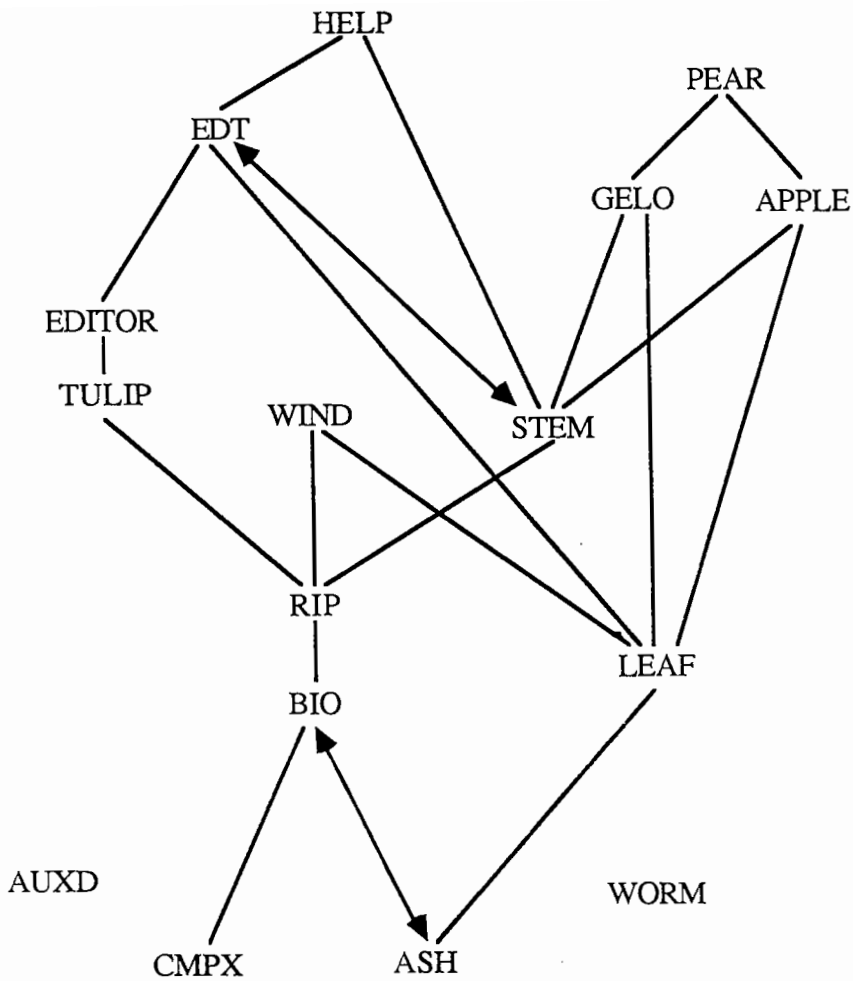


Figure 2: Interconnections of the various BWE components. Packages located higher in the diagram make use of the lower packages to which they are connected.

as opposed to event-based. Rectangular regions can be defined in any ASH window. Associated with each region is a routine to be called whenever an appropriate event occurs in the region. The programmer specifies the types of events the region should respond to. These can include various combinations of mouse transitions along with ranges of text and function keys. Overlapping or duplicate regions with different sets of events are allowed and handled with the more recent region being given priority. The X11 window system also supports application call-back methods, but this is only on a per window basis, and its event classes are not as specific as those in RIP.

The use of region-based input with call-back greatly simplifies applications. Most styles of menu buttons as well as permanent buttons on the screen are naturally defined as rectangular regions in a window. The use of call-back means there is no need for a control loop within the application to determine how to interpret each event and then takes the appropriate action. Rather, RIP acts as the distribution mechanism. This approach allows applications to be constructed modularly, with each module registering its regions separately. RIP is used extensively by STEM and other BWE components. Moreover, the selective call-back philosophy is repeated in the other components.

Additionally, RIP provides facilities for graphical input. Routines for input of points, lines, boxes, and polygons are supported, as is mouse tracking and rubber-banding. Textual input of an integer, a floating point number, or a text string is also provided.

3.2 STEM – Menuing

STEM is the component of BWE that adds menu capabilities to the Environment. STEM provides four types of menus: list type menus with automatic scrolling and layout, static menus that allow the user to select combinations of states, Macintosh-style pull-down menus, and hierarchical pop-up menus. It also provides automatically formatted scroll bars and powerful dialog boxes. STEM includes facilities for dynamically disabling or removing interactors such as dimming a menu item or not highlighting it as the mouse passes over.

Much of STEM's power results from the use of table-based descriptions for defining menus. As opposed to describing menu contents through parameters in procedure calls, STEM provides a significantly simpler programming interface. Moreover, the use of call back routines for menu buttons provides a consistent and modular input interface.

Particularly interesting is the string-based specification technique used for dialog boxes. The contents of a dialog box in STEM is specified by a string (or an array of strings) containing formatting commands and button definitions. Formatting commands include simple text output, font changes, justification options, and color information. Five types of interactive buttons exist: push buttons, named buttons, text regions, lists, and edit regions. The two button types can be used for either Boolean values or for multiple choice values (radio buttons). Named buttons for

explicitly accepting or cancelling the dialog are also provided. List buttons use the STEM list menu facilities to allow the user to select an element from a list. Arbitrary lists are allowed and STEM automatically provides scroll bars and a multi-column display as appropriate. Edit regions use the EDT base editor to allow the user to enter any number of lines of text.

A dialog box is invoked by passing the definition string and a set of pointers to variables. If the user accepts the dialog box rather than cancelling it, the new values are stored in the passed variables. An example of a dialog box generated by STEM is shown in Figure 3. The upper part of the figure shows the actual dialog box while the lower part shows the array of strings used to create it.

3.3 LEAF – Window Geometry

LEAF is the window layout and geometry component of BWE. LEAF sets up and maintains ASH windows through the use of a table-based, static, parameterized data structure. A large number of window attributes can be modified through LEAF via the specification of a window type and a window geometry.

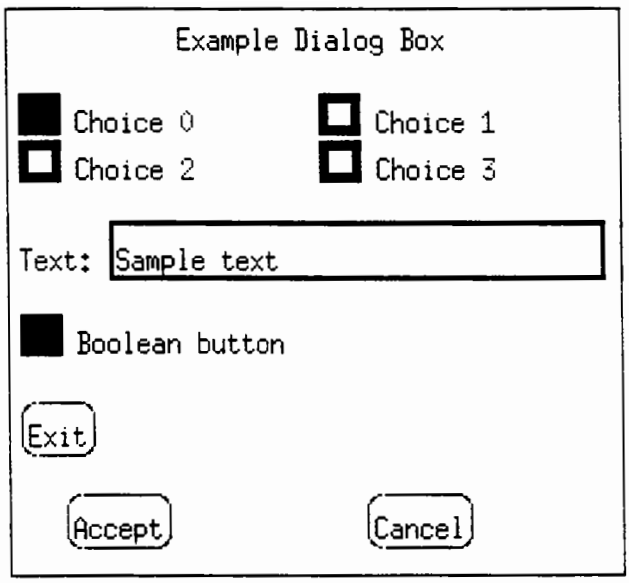
Window types provided by LEAF include simple windows, text regions, STEM menu regions, and filled boxes. Attributes such as a window's title, placement, sensitivity, and visibility can be set. Vertical or horizontal arrays of these windows are also available. Window geometry options for placing corners include absolute and relative positioning and sizing, "next-to" and "same" placement, along with relative component placement. An example of the use of LEAF and its result is shown in Figure 4. Here the main window has been subdivided into a number of regions for buttons and the title around its border. The center region can then be passed to another part of the application. This window is similar to one of the options for window management provided by WIND.

The table-based window layout with the amount of flexibility provided by LEAF is quite unique. It provides a limited but useful form of constraint-based layout. We discuss other toolkits with more advanced constraint facilities in Section 5 on related work.

3.4 EDT – Base Editor

EDT is the base editor of BWE. While base editors are not unique, we felt there was a strong need to have a complete and powerful editor as an integral part of the environment. Many of the other BWE components utilize EDT edit pads for "on-the-fly" text display and manipulations.

EDT contains features comprising those most often seen in modern text editors. In addition to the typical edit commands, facilities for executing commands through Macintosh-style pull-down menus and scroll bars exist. The mouse is fully integrated with EDT allowing cursor placement and selection or deletion of character, word, line, and paragraph units. The editor can be customized for a particular user or a




```
static  char *  dialog[] = {
   "%CExample Dialog Box\n",
   "%0.0o Choice 0%M%0.1o Choice 1",
   "%0.2o Choice 2%M%0.3o Choice 3\n",
   "Text: %1.32t\n",
   "%2o Boolean button\n",
   "%3'Exit'a\n",
   "   %a%M   %c",
   0
};
```

Figure 3: An example dialog box generated by the STEM package and the array of strings used to create it.

```
#define ROOT_WINDOW      0
#define USER_WINDOW      1
#define UL_REGION        2
#define TITLE_REGION     3
#define UR_REGION        4
#define LL_REGION        5
#define LOWER_REGION_1   6
#define LOWER_REGION_2   7
#define LR_REGION        8

static LEAF_DATA leaf_example[] = {
   LEAF_ROOT(NULL),
   { UL_REGION, LEAF_TYPE_TEXT|LEAF_TYPE_SENSE_FLIP,
        { LEAF_COORD_LX_LINE, LEAF_COORD_SIZE(ICON_HEIGHT),
             LEAF_COORD_SIZE(ICON_WIDTH), LEAF_COORD_TY_LINE },
        handle_move_btn, (int) MOVE_CHAR },
   { TITLE_REGION, LEAF_TYPE_TITLE,
        { LEAF_COORD_NEXT_LINE(UL_REGION), LEAF_COORD_SAME(UL_REGION),
             LEAF_COORD_RX_LINE, LEAF_COORD_TY_LINE },
        NULL, (int ) "Here is the title" },
   { USER_WINDOW, LEAF_TYPE_WINDOW|LEAF_TYPE_UPPER_LEFT|LEAF_TYPE_INVISIBLE,
        { LEAF_COORD_LX_LINE, LEAF_COORD_NEXT_LINE(LL_REGION),
             LEAF_COORD_RX_LINE, LEAF_COORD_NEXT_LINE(UL_REGION) },
        NULL, NULL },
   { LL_REGION, LEAF_TYPE_TEXT|LEAF_TYPE_SENSE_FLIP,
        { LEAF_COORD_LX_LINE, LEAF_COORD_BY,
             LEAF_COORD_SIZE(ICON_WIDTH), LEAF_COORD_SIZE(ICON_HEIGHT) },
        handle_size_btn, (int) SIZE_CHAR },
   { LOWER_REGION_1, LEAF_TYPE_TEXT|LEAF_TYPE_SENSE_FLIP,
        { LEAF_COORD_NEXT_LINE(LL_REGION), LEAF_COORD_BY,
             LEAF_COORD_TEXT, LEAF_COORD_SAME(LL_REGION) },
        handle_remove_btn, (int) "Remove" },
   { LOWER_REGION_2, LEAF_TYPE_TEXT|LEAF_TYPE_SENSE_FLIP,
        { LEAF_COORD_TEXT, LEAF_COORD_BY,
             LEAF_COORD_NEXT_LINE(LR_REGION), LEAF_COORD_SAME(LL_REGION) },
        handle_push_btn, (int) "Push" },
   { LR_REGION, LEAF_TYPE_TEXT|LEAF_TYPE_SENSE_FLIP,
        { LEAF_COORD_SIZE(ICON_WIDTH), LEAF_COORD_BY,
             LEAF_COORD_RX_LINE, LEAF_COORD_SAME(LL_REGION) },
        handle_size_btn, (int) SIZE_CHAR },
   LEAF_END
};
```

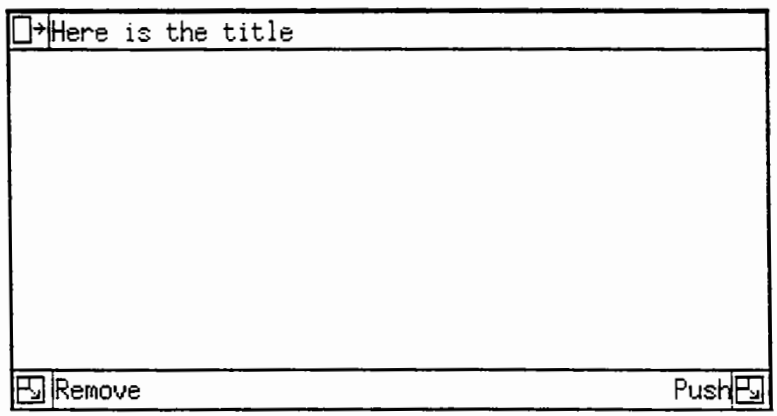


Figure 4: The definitions used to create a LEAF window and the resulting window image.

particular application in terms of key bindings, the contents of pull-down menus, the definitions of commands, and the default parameter settings.

The extensibility of EDT is established by allowing new commands to be defined through external, dynamically loaded C programs. Commands can also be defined in this language as specializations of existing commands. Thus, the usual case is to define a generic version of the command to be loaded in and then to define the more useful cases as specializations. The definition language also allows hook routines to be defined for indentation and selection.

3.5 HELP – A Common Help Facility

The HELP facility of BWE provides a consistent interface for displaying hierarchical help information to the user. By pressing the "Help" key, the user receives help information determined by her/his current location in the window hierarchy and RIP regions. The information is displayed in an EDT edit region annotated by STEM buttons and menus.

Help information files can be structured in either text files or directories. If the help search path is a directory, then each file within the directory is assumed to be a subitem, and the search continues recursively. Within text files, facilities for nesting and cross-referencing information are available.

In addition to hierarchical access to help files, the help facilities can provide mouse-sensitive help, offering a window that gives a brief description of the use of the mouse buttons and that is dynamically updated as the mouse moves.

3.6 GELO – Data Structure Display

The GELO, APPLE, and PEAR components of BWE help display complex program and data structures in their classical notations with a minimum of effort. Strongly typed data structures are displayed as picture objects of types data, tile, layout, arc, and empty. The display mechanism differs from many others in that diagrams are not described in a world coordinate space, but by giving a set of topological constraints for their views.

The three components interact as follows: GELO manages the specification and display of program and data structures. APPLE allows a user to design and customize the way that a particular data structure will appear by providing mechanisms for defining the mapping between the data structure and the GELO structures. (More than one mapping is allowed.) Finally, PEAR uses these mappings to allow the user to edit the structures' display, thereby modifying the underlying data structures. Figure 5 shows a GELO display (using PEAR) of a user's tree data structure in the upper half window and the APPLE description of how to draw the structure in the lower window.

The methods and functionality of these components are derived from work in the GARDEN graphical programming system developed by Reiss. For more detailed

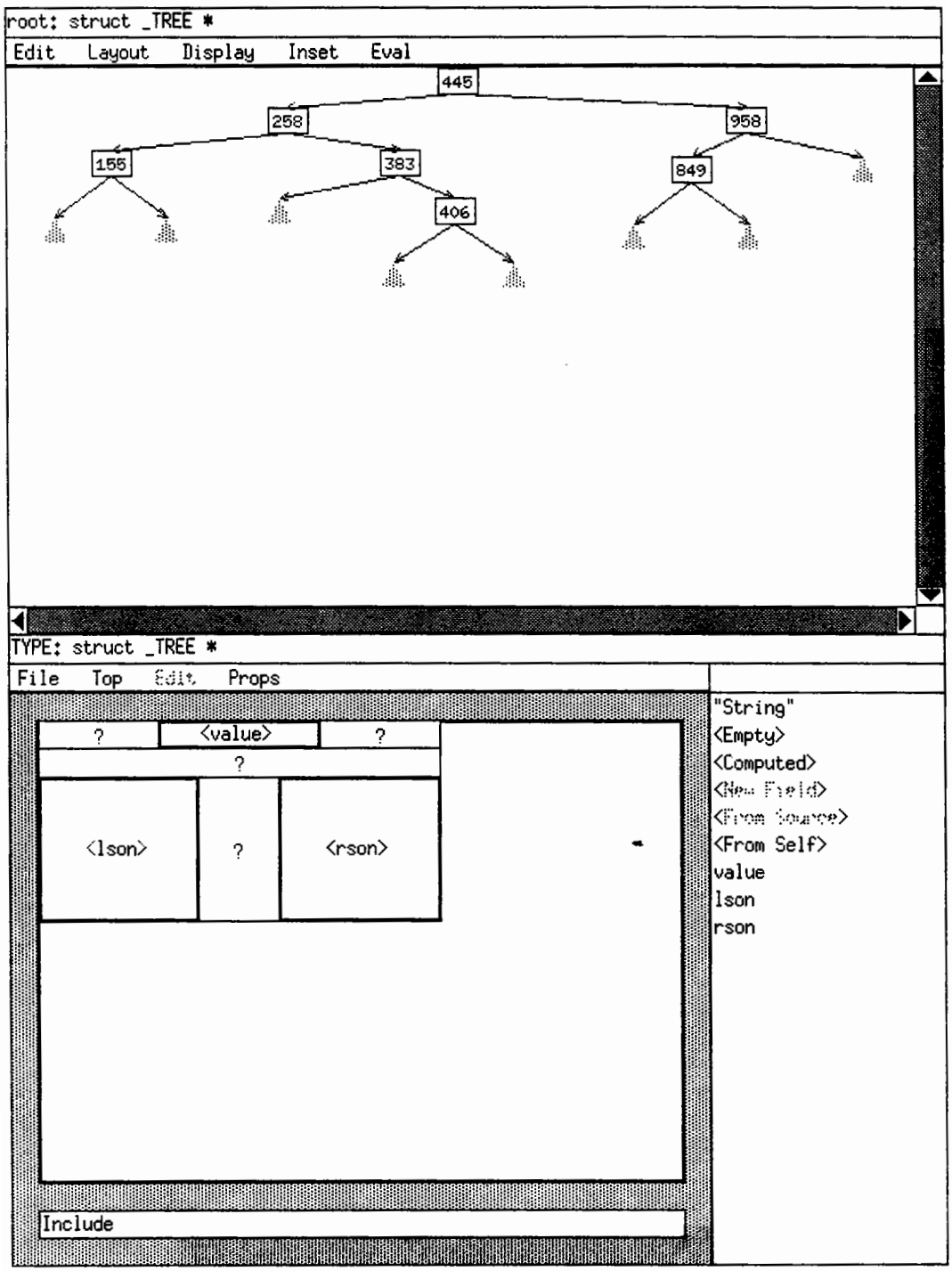


Figure 5: A GELO display of a user's tree data structure using PEAR, along with the APPLE description of how to draw the structure.

information on these components, see [19, 18].

4 Related Research

For an excellent review of the current state-of-the-art in user interface development systems, see [13]. Here, we will provide a brief description of and pointers to interface design toolkits of note. Two of the best known examples, the X11 toolkit in the X Window System[11] and the MacApp application interface design toolkit for programming in the Macintosh toolbox[24, 1], provide lower-level routines within an object-oriented programming paradigm. These toolkits are wide reaching and commonly used, but they are often characterized as being complex and difficult to learn. The Andrew toolkit[14] also provides an object-based paradigm that includes simple objects such as text, buttons, and scroll bars, as well as more complex objects such as tables, spreadsheets, and rasters. The Andrew toolkit was designed to be window system independent, and it has been used to implement interfaces to applications such as editors, a mail system, and a help system. Two toolkits that utilize constraints to manage interface appearance are Grow[3] and Coral[26]. Grow provides a hierarchy of graphical objects that support interobject relationships. Coral also provides graphical objects and constraints, as well as active values, constraints on lists or groups of objects, and a simpler declarative specification. The InterViews graphical user interface toolkit[10] provides an object-oriented environment in C++ consisting of interactor, graphic, and text base classes. These classes can be combined into compositions primarily through a method similar to boxes and glue in TEX[8]. The FormsVBT graphical interface builder[2] uses a two view design method: one view consists of a textual description of an interface in a special purpose language and the other view consists of a direct manipulation, interactive graphical editor. Changes made to the interface in one view are reflected in the other view as well. The language for describing the interface uses TEX-style syntax in a similar manner to InterViews, but within LISP-like symbolic expressions.

5 Conclusion

All the features of the Brown Workstation Environment described in this paper are up and running on a network of SUN and DEC workstations. BWE is widely used by a variety of clients: graduate students use it as a test-bed for implementing the ideas in their research, and it also serves as a backbone support environment for students in systems design courses. Three major systems research projects currently underway at Brown, the GARDEN conceptual programming environment[16], the FIELD program development environment[17], and the TANGO algorithm animation system[25], utilize BWE as a base environment. BWE is also available for distribution.

Future plans for BWE include rewriting GELO/APPLE/PEAR which were originally designed as part of the GARDEN graphical programming system. The rewrite would provide the user more flexibility in designing the look and layout of the data structure presentations. We are also developing a user interface builder, PINE, that would interactively design interfaces by demonstration something like that done in Peridot[12], Graffiti[4], by Cardelli[7], and OSU[9]. PINE would allow programmers to create, in a very simplified manner, the types of interface features currently available through STEM.

Acknowledgments
BWE has been, and still is, a large, cooperative development effort here at Brown. In fact, a report already exists describing the early form of the toolkit[20]. Many people have made significant contributions to the evolution of BWE. The authors wish to acknowledge the contributions of Joe Pato, Marc Brown, John Bazik, Eric Golin, Rob Rubin, Stefan Tucker, and Kevin Brophy. Tom Doeppner designed the THREADS lightweight process package used by BWE.

References

[1] Apple Computer. *The MacApp Reference Manual*, 1987.

[2] Gideon Avrahami, Kenneth P. Brooks, and Marc H. Brown. A two-view approach to constructing user interfaces. In *SIGGRAPH '89*, Boston, MA, August 1989.

[3] Paul S. Barth. An object-oriented approach to graphical interfaces. *ACM Transactions on Graphics*, 5(2):142–172, April 1986.

[4] Michel Beaudouin-Lafon and Solange Karsenty. Prototyping user interfaces for applications depicted by graphs. In *Proceedings of the 21st Hawaii International Conference on System Sciences*, pages 436–445, Hawaii, January 1988.

[5] Marc H. Brown and Robert Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2(1):28–39, January 1985.

[6] Brown University Computer Science Department, Providence, RI. *The Brown Workstation Environment Programmer's Manual*, 5.1 edition, December 1988.

[7] Luca Cardelli. Building user interfaces by direct manipulation. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pages 152–166, Banff, Alberta, Canada, October 1988.

[8] Donald E. Knuth. *The TeXbook*. Addison Wesley, Reading, MA, 1984.

[9] T.G. Lewis et al. Prototypes from standard user interface management systems. *IEEE Computer*, 22(5):51–60, May 1989.

[10] Mark A. Linton, John M. Vlissides, and Paul R. Calder. Composing user interfaces with InterViews. *IEEE Computer*, 22(2):8–22, February 1989.

[11] J. McCormack and Paul Asente. An overview of the X Toolkit. In *Proceedings of the 1988 ACM SIGGRAPH Symposium on User-Interface Software*, pages 46–55, Banff, Alberta, Canada, October 1988.

[12] Brad A. Myers. Creating dynamic interaction techniques by demonstration. In *Proceedings of the 1987 Conference on Human Factors in Computing Systems and Graphic Interface*, pages 271–278, Toronto, Canada, April 1987.

[13] Brad A. Myers. User-interface tools: Introduction and survey. *IEEE Software*, 6(1):15–23, January 1989.

[14] Andrew J. Palay et al. The Andrew Toolkit—An overview. In *Proceedings of the USENIX Winter Conference*, pages 9–21, Dallas, TX, February 1988.

[15] Steven P. Reiss. PECAN: Program development environments that support multiple views. *IEEE Transactions on Software Engineering*, 11(3):276–285, March 1985.

[16] Steven P. Reiss. Working in the GARDEN environment for conceptual programming. *IEEE Software*, 4(6):16–27, November 1987.

[17] Steven P. Reiss. Integration mechanisms in the FIELD Environment. Technical Report CS 88-18, Brown University, Providence, RI, October 1988.

[18] Steven P. Reiss, Scott Meyers, and Carolyn Duby. Using GELO to visualize software systems. In *UIST '89*, Williamsburg, VA, October 1989.

[19] Steven P. Reiss and Joseph N. Pato. Displaying programs and data structures. In *Proceedings of 20th Hawaii International Conference on System Sciences*, January 1987. Also available as Brown Univ. Technical Report CS 86-19.

[20] Steven P. Reiss, Joseph N. Pato, and Marc H. Brown. An environment for workstations. In *Proceedings of the IEEE Conference on Software Tools*, pages 112–117, April 1985. Also available as Brown Univ. Technical Report CS 84-03.

[21] J. Rosenberg et al. UIMSs: Threat or menace? In *Proceedings of the 1988 Conference on Human Factors in Computing Systems and Graphic Interface*, pages 197–200, Washington, D.C., May 1988.

[22] Robert V. Rubin, Eric J. Golin, and Steven P. Reiss. ThinkPad: A graphical system for programming-by-demonstration. *IEEE Software*, 2(2):73–78, March 1985.

[23] Robert W. Schiefler and Jim Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[24] Kurt J. Schmucker. MacApp: An application framework. *Byte*, 11(8):189–194, August 1986.

[25] John T. Stasko. *TANGO: A Framework and System for Algorithm Animation.* PhD thesis, Brown University, Providence, RI, 1989. Also available as technical report CS-89-30.

[26] Pedro Szekeley and Brad A. Myers. A user-interface toolkit based on graphical objects and constraints. *SIGPLAN Notices, OOPSLA '88*, pages 36–45, November 1988.