Edgar H. Sibley
Panel Editor

Many contemporary user oriented applications require a combination of attributes from both window and forms management systems. PANES was developed as a tool to fill this niche by providing a simple, yet flexible tool for both the moderately trained, as well as the more sophisticated programmer, while allowing the development of user friendly, modeless application software.

ANATOMY OF A COMPACT USER INTERFACE DEVELOPMENT TOOL

JACK W. STOTT and JEFFREY E. KOTTEMANN

The development of a highly interactive, information modeling system [8] necessitated the selection or construction of a user interface development tool (UIDT). Like many current applications, this modeling system processes symbolic expressions and text as well as traditional data. Further, the UIDT must address command and control protocols in such a way that user interfaces can be customized for particular applications. A search of the literature and available products uncovered several UIDT alternatives. These alternatives can be categorized in two groups: forms managers and window managers.

FORMS MANAGERS

Forms managers typically view the screen as a collection of data elements, for example, a screen form version of an employee record. They give the programmer the capability to define data input screens, validate input data, process basic commands, and display results. Forms managers can be grouped into three categories:

(1) *Generalized, embedded forms managers.* These are usually implemented as language extensions, either intrinsic to a language processor (an extension to the compiler) [6, 10] or as sets of extrinsic routines linked to programs at link edit time [1–3]. They usually include a set of functions to manage terminal input and output, and also some functions that validate a set of data types. These UIDTs aid programmers in the *construction* of interfaces for data-oriented applications.

(2) *Generalized, nonembedded forms managers.* These are front-end input processors (or filters) that collect input and place it in an intermediate file to be used by batch application programs. Examples of these can be seen in [7] and [11]. These UIDTs eliminate the need to construct user interfaces for each application.

(3) *Application-specific, embedded forms managers.* These are represented by those found in popular database management packages, for example, RBase and dBase. They function much like the generalized, embedded UIDTs, except that they can only be used in conjunction with the host packages. They may also be used as filters or front ends like the generalized, nonembedded UIDTs.

WINDOW MANAGERS

Window managers are usually implemented as part of an operating-system environment. Sometimes they are embedded in applications such as word-processing packages. In the operating-system environment, a window is a logical view port for each job in the multitasking environment; a screen is a physical window upon which each logical window is mapped. Usually, the end user can change the size and orientation of a logical window to the physical window. Typically, only one job is active at a time, such that *opening* a particular window is comparable to activating that job. In some windowing environments, the Macintosh, for instance, the operating system's window management routines are made available to application programmers. The programmers can then use windows in their applica-

tions for such purposes as pull-down menus and dialogue boxes. Dialogue boxes are, in essense, form windows in which the end user can set task parameters.

INTEGRATING CONCEPTS FROM BOTH WORLDS

From the perspective of an application programmer, two critical and contradictory aspects separate forms-oriented and window-oriented UIDTs: flexibility and complexity. Although forms managers are relatively easy to learn and use, their overall support for developing interactive applications is quite limited. Window managers are powerful tools, yet require assiduous training. For example, Macintosh programmers must know numerous internal details of the operating-system environment, the object-oriented programming paradigm, and debugging at the Motorola 68000 object-code level.

Our primary goal was to provide both forms oriented and selected window-oriented interface development support. The relative simplicity of forms oriented interface was to be maintained. The resulting UIDT, PANES, satisfies these basic requirements. PANES has been successfully used by moderately skilled programmers after limited training. It has shown itself flexible enough to be useful in a number of application areas that include office automation, knowledge-based expert systems, document management, and data processing. PANES was originally implemented in Pascal on a Digital VAX under VMS. Since then, it has been implemented on microcomputers under MS-DOS and in C under UNIX®. The basic nature of PANES is illustrated by the list of requirements it was designed to satisfy (see Figure 1).

MODEL APPLICATION

In order to motivate the general requirements and characteristics of UIDTs such as PANES, we will simulate an application. The example is a KWIC-indexed document-retrieval system that allows users to search for documents by entering a series of keywords to be used for matching document titles. Further, the user may specify, via the menu, that the keywords are to match the titles only if they appear (1) in any order, (2) in relative order, or (3) in strict consecutive order (e.g., the keywords *dolphin* and *person* will match the title "A Dolphin Is Smarter than a Person" only under the first and second ordering criteria). In addition to perusing a database of documents, the user may create a report that lists the current document information and insert comments within the report.

In order to make the application user friendly, function keys are assigned to the various application functions:

FK1: Activate[1] the next user I/O window;
FK2: Find the first or next matching document;
FK3: Print the current document information to a report; and
FK4: Open a text input window to let the user enter text to be written to the report—perhaps a report heading or comments about the current title.

After using FK4 the following functions become active:

FK4: (Used á second time) to confirm that the entered text should be printed to the report; and
FK5: To cancel the display of the comment.

The screens for the KWIC application are shown in Figures 2a and b. Notice that the screen is composed of several windows. Each window serves a different purpose and therefore behaves differently, as the following explains.

- Windows (1), (2), (4), (6), (8), and (10) of Figure 2a and window (12) of Figure 2b are label windows. Label windows display labels, or any text, that describe other windows or the entire screen. Once label windows are displayed, the application never allows the user to enter these windows.

(1) To provide window-oriented textual interfaces for individual applications, where windows could be assigned separate operational attributes and functions within each application

(2) To allow definitions of and provide management for data and text, as well as menu windows and commands

(3) To allow specification of the conditions under which the user interface system should return control to the application, including application-level command inputs as well as a number of status conditions, such as buffer overflows

(4) To hide the application-to-user-interface-system functional layers from users—that is, users should not need to give one command to exit from the dialogue manager and then another to initiate action in the application; per above, the dialogue manager must then know what commands should be passed up the line

(5) To provide predefined functions and control protocols for manipulating windows so that assorted applications would have consistent user interfaces

(6) *Yet*, to allow applications to easily override the user interface system's functions

(7) To be hardware independent, assuming only an asynchronous, 7-bit ASCII terminal with a minimum set of terminal control commands (e.g., cursor positioning) attached to an 8-bit per byte computer

(8) To provide these capabilities in a manner that makes the application development environment usable to moderately experienced programmers, while also allowing experienced programmers the utmost latitude to define "renegade" types of windows and to access low-level functions

FIGURE 1. Basic Requirements on a UIDS

UNIX is a registered trademark of AT&T Bell Laboratories.

[1] In this paper windows being *activated* means that they become ready to accept user input. Only one window can be active at a time.

(1) Document query system

(2) Input keywords:

(3) PERSON
DOLPHIN

(4) Word matching criteria:

(5) Any order in title
Relative order
Strict order

(6) Matching document ID:

(7) 0-12-3700035-60

(8) Matching document title:

(9) A Dolphin Is Smarter than a Person

(10) Abstract:

(11) Based on the volume of brain matter and the relative size of the medulla oblongata, the dolphin perhaps has advanced facility for some activities that are more advanced than those that humans may be able to accomplish. It seems

FIGURE 2a. Primary Screen for an Example Document Query Application

Document query system

Input keywords:

PERSON
DOLPHIN

Word matching criteria:

Any order in title
Relative order
Strict order

Matching document ID:

0-12-3700035-60

Matching document title:

A Dolphin Is Smarter than a Person

(12) Input user comment

(13) This article seems to be relevant to our current research. Please route to Bob.

FIGURE 2b. Secondary Screen for an Example Document Query Application

- Window (3) is a user window used to input keywords. Window (3) is active upon entry to the application or can be activated by use of the FK1 key. User windows are used for general user input and output. In this example, the application's database management system requires that the keyword tokens be entered in a list. Therefore, the application must ensure that each keyword is separated by a carriage return.
- Window (5) is a menu window used to specify the matching criteria. Window (5) can also be activated using the FK1 key. The user may use the up and down arrow keys to select an option, or hit A, a, R, r, S, or s. After some keywords are entered into window (3) and a selection is made in window (5), the FK2 key may be used to find titles that meet the given criteria.
- Windows (7), (9), and (11) are read-only windows used to display information on the current matching document. Window (7) displays the current document identifier; window (9), the current document title; and window (11), a summary of the document. Since the abstract in window (11) may be longer than fits on the screen, the user may use FK1 to select window (11) and then use the up and down arrow keys to scroll through the remainder of the abstract.
- Window (13) of Figure 2b is a user window used to accept a comment entry from the user to be written to the report. Note that windows (12) and (13) overlay windows (10) and (11). When the user strikes FK4, windows (12) and (13) appear, and window (13) becomes active, allowing the user to enter a comment.

In this application the windows used are not an exhaustive set of the different types definable by using PANES. Table I reviews these types and describes some other basic types of windows. The example shows a typical application for a UIDT system to handle. Basic requirements of the UIDT that surface here are (1) general interface, (2) user-oriented characteristics, and (3) programmer-oriented characteristics.

General Interface Requirements

Many applications, as shown in the example, require the UIDT to support *forms* whose *fields* act as windows. As seen in the example, the fields are multiple line fields (windows (9) and (11)) or field groups (window (3)). The UIDT must be able to handle fields that have separate operational attributes and functions. As in the example, windows for data (including multivalued fields), text, menus, and commands must be supported.

User-Oriented Requirements

Primary objectives of any user interface are ease of use and consistent behavior within families of applications (see [4]). Given a good UIDT, the latter requirement can be fulfilled by merely using the UIDT for all applications.

Second, the application-to-UIDT functional layers should be hidden from the users. In the example, for instance, the user should not need to give one command to exit the UIDT procedure and then give the FK2 command to initiate the application's find action. Rather, whenever FK2 is entered, a matching document should be immediately displayed. In the Smalltalk/Xerox Star [5] and later the Macintosh [9] environments, this concept is part of a philosophy termed *modeless operation*.

Programmer-Oriented Requirements

To reduce development cost, the UIDT should be easy to use, yet support many variations of data, text, and command windows. Also, the programmer must be able to easily define conditions under which the UIDT process should return control to the application, and the UIDT must also return the command input as well as any relevant status conditions. This requirement is related to the above user requirement. As the user strikes the FK2 command, the UIDT process must both return to the application and let the application know that it is returning because of the command.

TABLE I. Some Basic Window Types Supported by PANES

Type	Usage	Content loading	Special notes
Label (windows (1), (2), (4), (6), (8), (10), and (12) of Figure 2)	Displays labels for other windows or titles for a screen	Application loads at program initiation	User may not enter; may be highlighted or boxed
Read only (windows (7), (9), and (11) of Figure 2)	Displays help, program messages, or application documentation	Application loads at program initiation, or from files at or during run time	User may not alter; may be more text than fits in the window, so must allow scrolling
Menustatic (window (5) of Figure 2)	Displays alternatives for application execution	Application loads at program initiation	May be horizontal or vertical; user may use cursor keys to select
Menudynamic	Displays application alternatives for user's choice	Application or user loads during application execution	User may build a menu
User (windows (3) and (13) of Figure 2	Used for general user input and output	User input or application loads during execution from files or other windows	Input may be filtered to allow data edits

The UIDT must also be flexible enough to allow programmers to override or redefine any of the UIDT functions. For instance, given a UIDT function `clear current window`, if the application has a window with a large amount of critical text, the application must be able to override that function to ensure that the user does not accidentally erase the text.

The UIDT should be very easy to use for novice programmers, yet allow experts the utmost latitude in defining renegade window types, and perform actions at the lowest possible level. Finally, to make the system portable, the UIDT should be as hardware independent as possible.

INTERFACE CHARACTERISTICS AND REQUIREMENTS

Understanding what an application might require from a UIDT, the operations and control mechanisms supported by PANES can now be explained.

Editing in Windows

PANES provides two primary routines for applications use:

(1) `DISPLAYER`: displays the current contents of a window onto the screen; and
(2) `EDITOR`: performs the full-screen text input and editing in any window on the screen.

To the user, PANES appears similar to common, full-screen window-oriented text editors with such amenities as word wrap and function-key-controlled editing. `EDITOR` has predefined commands that allow the user to move the cursor up or down a line, move the cursor to the right or left by character or word, move the cursor to the top or bottom of the window text, find a string, delete a character or word, or clear the entire contents of a window. The application can bind the invocation of these functions to any alternate key or keys if desired.

Beyond these primary operations, the UIDT must also "know" something about each type of window defined by the application programmer. In PANES this definition involves the specification of restrictions on the allowable user input (including command keys) when the given window is active, the mapping of user input (again, including command keys) to other keys, and command and control passing protocols.

Restricting Input

In many cases, the application would like to enforce restrictions on the character set allowed as input to a given window. These restrictions often apply to data input; for example, in a window that may contain only integers, the UIDT system should restrict input to the characters `0-9` and – (minus sign). An application may also want to restrict the user's use of the commands embedded in the UIDT procedures; for example, the application may want to disallow the user's ability to easily erase the entire contents of a window. To enforce this, the application may restrict the use of the `Clear Window PANES EDITOR` command in that window. Finally, the application will want to restrict the input of all other characters and command keys not explicitly allowed in the given window. Other examples of input restrictions are

- *label windows*, which restrict all input (see windows (1), (2), (4), (6), (8), and (10) in Figure 2a);
- *read-only windows*, which allow only cursor movement commands and the find command, and restrict all other input (see windows (7), (9), and (11) in Figure 2a);
- *menu windows*, which allow only cursor characters and the first characters of menu entries, and restrict all other input (see window (5) in Figure 2a);
- *user windows used for strictly alphabetic names*, which allow the upper- and lowercase alphabet, and PANES text editor commands, but restrict all other input; and
- *user windows used for specialized data elements*, such as a part number that uses only the characters `A`, `D`, `J`, `Q`, and `0-7`, which allow only certain characters and PANES text editor commands, but which restrict all others.

Note that the programmer would also define which application-level commands are allowable in each window.

Mapping Character Input

The application may also want to specify a mapping, or conversion, of some inputs before they are processed by PANES. Input character mapping is used to (1) enhance user friendliness; (2) implement a more active form of character restriction; (3) cope with differences in terminal environments; and (4) modify the behavior of PANES, specifically, the editing commands. These uses are exemplified in the following examples:

- Mapping restricted characters to related legal characters is utilized to improve user friendliness. For instance, in a window used for eliciting a columnar list of items in which each token entered must be separated by a carriage return, the programmer might specify that spaces be converted to carriage returns. This feature is used in window (3) of Figure 2a.
- A similar function is mapping the letter `o` to the number `0` for integer-type windows. This example shows the notion of active character restriction. The letter `o` is actually a restricted letter; however, instead of merely ignoring the input, the application tells PANES to assume what the user really meant to input.
- Read-only windows may map the left-arrow character to the up-arrow character, and the right-arrow, carriage-return, and line-feed characters to the down-arrow character, in order to implement a more flexible cursor movement.
- In terminal environments where there is both a `DELETE` and `BACKSPACE` key, the application could map both of them to the backward character delete function.

- Modifying the PANES EDITOR commands can be used to make the editor behave as a familiar word processor. For example, control-X may be mapped to DOWN_ARROW to emulate WORDSTAR's cursor movement scheme.

Basic Control Mechanisms

The control passing mechanism between the UIDT process and the application should be transparent to the user (or appear modeless). Since the UIDT receives all inputs, even application commands, many UIDT systems require the user to give one command to exit the UIDT level and then another to direct the application. A better control mechanism would allow the UIDT process to know the application commands, and upon receiving one, return control to the application, passing back the command. Upon regaining control, the application would then act on the command.

To implement such a control mechanism, the application defines a collection of keyboard commands that are active at the application level. Then as the user, who always appears to be active inside one window or another, issues a keyboard command (strikes a control or function key), the PANES EDITOR determines if control should be returned.

THE CURRENT IMPLEMENTATION OF PANES

In developing PANES we desired a simple, unified window definition scheme that would capture the behavioral characteristics of the various logical window types given above. This is accomplished via definition of input control tables.

Window Definition via the Input Control Table

Thus far, window types to be supported include labels; various types of menus, and an assortment of read-only and user input windows capable of restricting and mapping inputs. Further, the application must be able to define its command keys, as well as override or redefine PANES EDITOR commands when desired. The definition of these various aspects is simplified and unified when one considers that an entered character or sequence (e.g., an escape sequence generated by a function key) can be classified into one of four mutually exclusive sets:

(1) a command to return to the application;
(2) a restricted character, to be ignored;
(3) a command to PANES; or
(4) a text character to be used as input to the window.

Also, the application may require an input to be converted before determining which set the input belongs to.

For implementation, each window's definition includes a table that maps and then categorizes each input. In PANES this table is called the INPUT_CONTROL_TABLE (ICT). The ICT is a character array, 256 bytes long, containing a value for each possible keyboard input. (Although the ASCII character set defines only 128 characters, a low-level routine maps function-key character sequences and other key sequences to the high-order values, 128–255. Control sequences, that is, function keys, may then be treated as single-character input by PANES and the application.) Each element in the ICT corresponds to one of 256 possible input characters or character sequences. Each element position is given a value:

(1) RETURN_CODE (ASCII 255), if the input indicates return of control (i.e., if ICT[ord(input)] = RETURN_CODE, return control and the input to the application);
(2) RESTRICTED_INPUT (ASCII 0), if the input is illegal for this window;
(3) the alternate value that an input is to be mapped to (if the input is to be mapped); or
(4) the same value as the input, if the input is to be put into the window or is a PANES-level command to be respected.

Using this simple map table scheme, all of the window types and requirements previously discussed can be defined and implemented. For example, the ICT for a read-only window would

- map the application commands to RETURN_CODE;
- map the UP_CURSOR and DOWN_CURSOR characters to themselves;
- possibly map other characters to UP_CURSOR and DOWN_CURSOR (e.g., LEFT_CURSOR to UP_CURSOR, RIGHT_CURSOR to DOWN_CURSOR, CARRIAGE_RETURN to DOWN_CURSOR); and
- map all other characters to RESTRICTED_INPUT.

The ICT for a menu window would

- map the application's commands to RETURN_CODE;
- map the first character of each menu entry to themselves, or to RETURN_CODE (the first method requires the user to enter a choice and then hit an additional command to initiate action; the second only requires the user to hit the first character of the command to initiate action);
- map the UP_CURSOR and DOWN_CURSOR characters to themselves;
- possibly map other characters to UP_CURSOR and DOWN_CURSOR (e.g., LEFT_CURSOR to UP_CURSOR, RIGHT_CURSOR to DOWN_CURSOR, CARRIAGE_RETURN to DOWN_CURSOR); and
- map all other characters to RESTRICTED_INPUT.

For a text input window, the ICT would

- map the defined return control characters to RETURN_CODE;
- map all PANES commands and printable characters to themselves; and
- map all other nonprinting characters not used as commands to RESTRICTED_INPUT.

Any of the previously mentioned variations can also be implemented. If a window should only receive numeric input, all alphabetic and punctuation characters

```
RETURN_FLAG := false;
BUFFER_OVER_OR_UNDER_FLOW := false;
repeat
  IN_CHAR := GET_CH;
  MAPPED_CHAR := INPUT_CONTROL_TABLE[IN_CHAR];
  if MAPPED_CHAR = RETURN_CHAR
    RETURN_FLAG := true  {returns control to
      application}
  else if MAPPED_CHAR = RESTRICTED_CHAR
    BEEP  {an illegal character for this window}
  else if MAPPED_CHAR in [EDITOR_COMMANDS]
    PROCESS_EDITOR_COMMAND;
      {These commands include moving the cursor up
       or down a line, moving the cursor up or down
       a window, moving the cursor right or left a
       character, moving the cursor right or left a
       word, moving the cursor to the top or bottom
       of the window or text, finding a string,
       deleting a character or word, and clearing
       the entire contents of a window.}
  else
    ADD_CHARACTER_TO_WINDOW;
  if BUFFER_OVER_OR_UNDER_FLOW and
      (RETURN_ON_BACK or RETURN_ON_FORWARD) then
    RETURN_FLAG := TRUE;
until RETURN_FLAG;
```

FIGURE 3. EDITOR Top-Level Algorithm for Processing Input with the ICT

```
(* Static memory buffer descriptors *)
first_buffer_position       : integer;
last_buffer_position        : integer;
maximum_logical_lines       : integer;

(* Dynamic memory buffer descriptors *)
current_buffer_position     : integer;
current_end_of_text         : integer;
current_logical_lines       : integer;

(* Static screen partition descriptors *)
top_row                     : integer;
left_column                 : integer;
number_of_rows              : integer;
number_of_columns           : integer;

(* Dynamic screen partition descriptors *)
cursor_row                  : integer;
cursor_column               : integer;

(* DISPLAYER control parameters *)
draw_box                    : boolean;
is_Vmenu                    : boolean;
is_Hmenu                    : boolean;
null_character              : char;

(* EDITOR control parameters *)
reset_to_top                : boolean;
return_on_back              : boolean;
return_on_forward           : boolean;

input_control_table         : packed
                              array[0..255]
                              of char;
```

FIGURE 4. Window Description Parameters for PANES

would be mapped to RESTRICTED_INPUT. If a window should be a columnar list, SPACE would be mapped to CARRIAGE_RETURN.

Also, all PANES commands can be redefined or disabled by changing their values in the ICT. This allows substantial flexibility for the application programmer to modify PANES to resemble any environment with which the users are familiar. PANES functions can be modified by experienced programmers as follows: The commands of issue are made to map to RETURN_CODE; and application-specific routines are written and called when these commands are entered. Figure 3 shows the main procedure of the PANES EDITOR.

Miscellaneous Window Definition Parameters

In PANES the memory buffer is stored as a character array (in our Pascal implementation, PACKED ARRAY[n..m] OF CHAR). Each application usually defines one large memory buffer to hold the contents of all windows. (Although not required by PANES, using one buffer eliminates the need to declare and manage large numbers of different buffers.) DISPLAYER displays text and data from this buffer, and EDITOR manages the placement of data and text into the buffer as well as managing control functions.

The two types of parameters that describe the memory buffer and screen partitions are the static position indicators and the dynamic status indicators. The first describes where in the buffer and on the screen the

window resides, and the second describes the current state of the window. See Figure 4 for a list of these parameters, and Figures 5a and b for examples of their use.

In addition to the memory and screen parameters, and the ICT, a number of other parameters control miscellaneous aspects of PANES. They include the Boolean variables that determine whether or not to draw a box around a window, highlight the current line (used mainly for menus), reset the cursor to the top of a window upon (re)entry, and return control if the user moves past the beginning or end of the text of the window. These last two parameters are mainly used when there is more information than fits in the buffer space. Finally, a parameter specifies the character used to display instead of using spaces (e.g., underscores to indicate empty space).

THE PANES SUPPORT TOOLS

Window definition and instantiation involve loading text such as label values in buffers, setting and changing parameters, and initializing ICTs. To aid the application programmer, tools were created to eliminate the tedium connected with defining windows, storing their definitions, and initializing interfaces. The primary support tool is the PANES SCREEN_DEFINER. This tool allows the application programmer to interactively prototype a screen layout. SCREEN_DEFINER acts very much like an interactive editor—indeed it uses EDITOR and DISPLAYER—to edit existing and allow full-screen creation of new screen layouts.

From the above descriptions of parameters and ICTs, it would seem that this definition is a time-consuming process. Using the *concept* of logical window types described under "A Model Application," however, SCREEN_DEFINER can set the values of many of the window parameters and create an initial ICT based on the *logical* window type—label, menu, read-only, or user. The programmer may then alter any of these default values to add any of the variations discussed earlier.

The output of SCREEN_DEFINER is Pascal code for the declarations and procedures that define and initial-

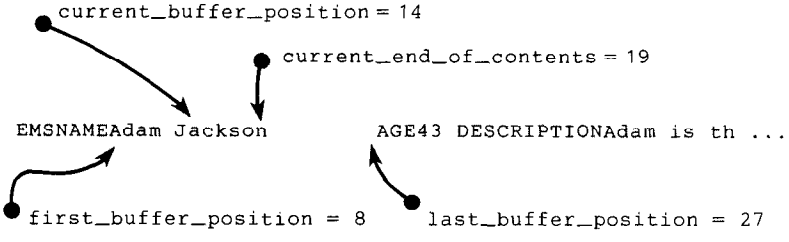

FIGURE 5a. Values of Memory Buffer Parameters for a Sample Window

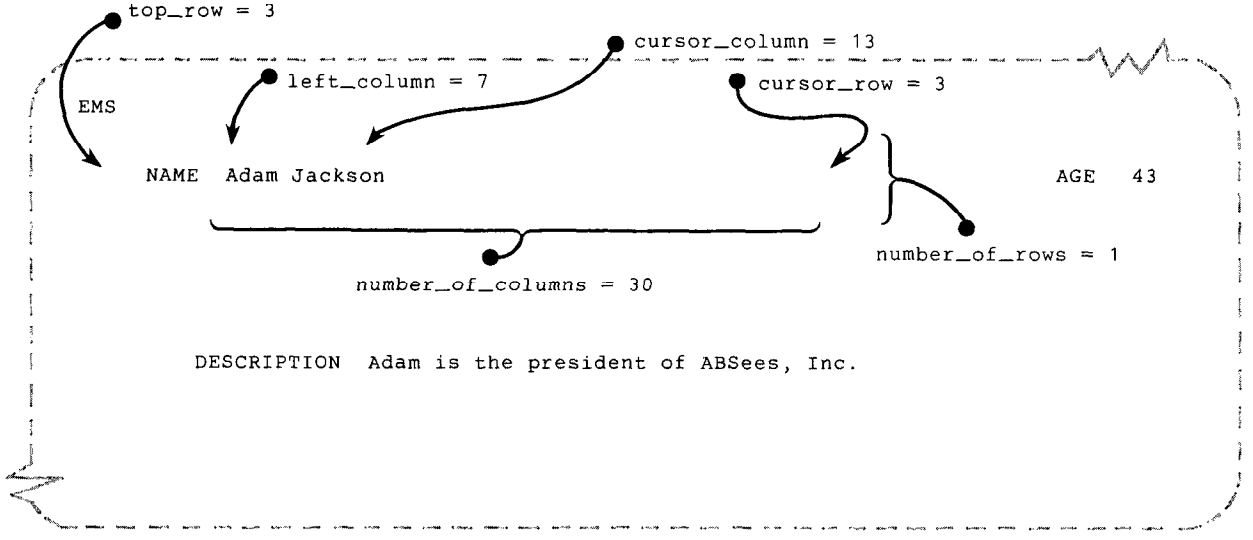

FIGURE 5b. Values of Screen Partition Parameters for a Sample Window

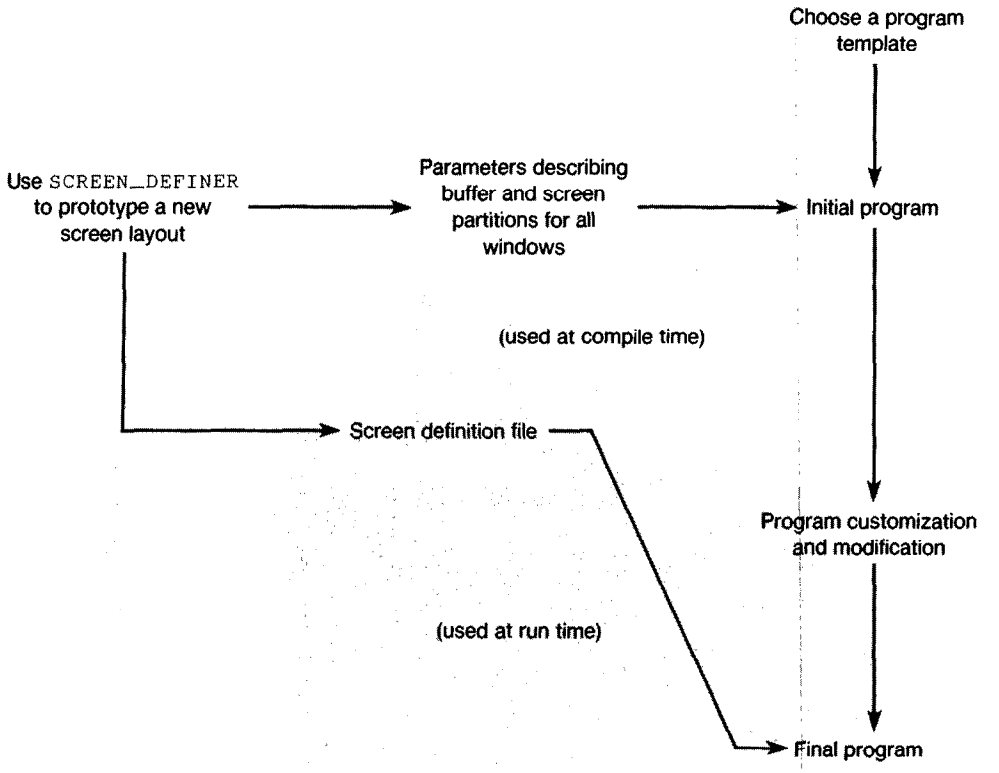

FIGURE 6. Using the PANES Support Tool Environment

```
begin  {main program}
  DONE := false;
  DISPLAYER(1, 11, WINDOW_PARAMETERS, WINDOW_BUFFER);
  CURRENT_WINDOW := 3;
  repeat
    EDITOR(WINDOW_BUFFER, WINDOW_PARAMETERS[CURRENT_WINDOW],
           RETURN_STATUS, RETURN_INPUT);
    case RETURN_INPUT of
      FK1: Set CURRENT_WINDOW to next enterable window;
      FK2: FIND_FIRST_NEXT;
        {This procedure uses the contents of windows (3) and (5) to
         search the database and fill windows (7), (9), and (11).}
      FK3: PRINT_DOC_INFO;
      FK4: begin
        DISPLAYER(12, 13, WINDOW_PARAMETERS, WINDOW_BUFFER);
        EDITOR(WINDOW_BUFFER, WINDOW_PARAMETERS [13], RETURN_STATUS,
               RETURN_INPUT);
        case RETURN_INPUT of
          FK4: PRINT_DOC_INFO;
            {This procedure prints window (13) to a report.}
          FK5: {null};
        end;
        ERASE_WINDOWS(12, 13, WINDOW_PARAMETERS);
        DISPLAYER(10, 11, WINDOW_PARAMETERS, WINDOW_BUFFER);
        end;
      CONTROL_E: DONE := TRUE;
    end;
  until DONE;
end;  {main program}
```

FIGURE 7. The Main Procedure for an Example Document Query Application Using PANES

What the application program does:	*What the user does:*
Program initialization; DISPLAYER displays windows (1)–(11) on the terminal screen;	
Activate window (3) by setting CURRENT_WINDOW to 3 and calling EDITOR; wait for user input . . .	Input several keywords into window (3); Use FK1 to activate window (5).
Because FK1 is defined as a return character, EDITOR returns control to the main procedure; control is passed to the FK1 label of the CASE statement where CURRENT_WINDOW is changed to 5 and EDITOR is called again; wait for user input . . .	Select matching criteria by using cursor keys or entering the first character of the desired choice; use FK2 to find the first document that meets the criteria.
Again, as FK2 is defined as a return character, EDITOR returns control to the main procedure; control is passed to the FK2 label of the CASE statement where a database query routine is called to get the first title and insert it, to get the document ID and abstract into their respective windows, and to call DISPLAYER to display their new contents; EDITOR is called again; wait for user input in window (5) . . .	Change the matching criterion; use FK2, repeating the last program step, or use FK1 twice.
CURRENT_WINDOW is now 11; wait for user input . . .	Use the cursor keys to scroll up and down through a large abstract; finally use FK1, FK2, FK3, or FK4, or use FK1 twice.
Using FK1 twice will make window (3) active again . . .	Use the EDITOR's editing capabilities to change the current keywords; use FK1 or FK2, or use FK4.
Control is returned to the main procedure, and the FK4 label of the CASE statement is executed where DISPLAYER overwrites window (10) and (11) with windows (12) and (13), and saves the CURRENT_WINDOW in SAVE_WINDOW; EDITOR is called with window (13) active; wait for user input . . .	Enter text of a comment to be included in the report; use FK4 again.
Control is returned to the main procedure, and the FK4 procedure is executed to print comments to the report; DISPLAYER overwrites windows (12) and (13) with windows (10) and (11); EDITOR is called with the SAVE_WINDOW active; wait for user input . . .	Use FK3.
Control is returned to the main procedure, and the FK3 procedure is executed; the current title, document ID, and abstract are written to a report file; Control is returned to EDITOR with the CURRENT_WINDOW still active; wait for user input . . .	Use FK1 or FK2.

FIGURE 8. Walk-Through of an Example Document Query Application

ize parameters. SCREEN_DEFINER also produces a screen definition file so that a window definition may be edited later. Finally, a summary report of the screen definition is prepared. The role of SCREEN_DEFINER is summarized in Figure 6. Another version of SCREEN_DEFINER is application callable and will store parameter values in definition files. This feature allows applications to bind screen definitions at run, rather than compile, time. Observe, however, that, since applications can manipulate the definitions during execution, the definitions are never strictly bound.

At a lower level, application programmers have complete access to the window definition record and the library of routines upon which PANES is built. Though application programmers need know very little about the window definition parameters to use PANES, the parameters are directly available to the application, allowing the application to dynamically change attributes of the window at any time. This approach gives programmers both ease of use and a high degree of flexibility. The PANES software library also gives the programmer access to a multitude of string and screen handling tools that can be used to manipulate the buffer contents.

THE EXAMPLE PANES APPLICATION REVISITED

Using the main line code in Figure 7, the operation of the example application given under "A Model Application" will be explained.

After initialization procedures, the application, using DISPLAYER, displays the screen as shown in Figure 2a, and then activates window (3) by calling EDITOR with CURRENT_WINDOW equal to 3. The user may now enter one or more keywords. Using FK1 the user moves to window (5). Since FK1 is defined as a return character for window (3), EDITOR returns control to the applica-

tion, which examines RETURN_CHARACTER in the **case** statement and changes the currently active window to window (5). The user may now enter a matching criterion, or use the default already shown in the window. The user now hits FK2, the command to find the first document title. Again, since FK2 is defined as a return character, control is returned to the application that examines RETURN_CHARACTER in the case statement and calls the FIND_FIRST_NEXT application procedure. This procedure finds the first document, fills in the buffer area for windows (7), (9), and (11), and calls DISPLAYER, which displays these windows. Control returns back to the application, which calls EDITOR, still with window (5) active. The user may now hit FK1 once or twice to activate window (9) or (11), respectively, and scroll through a lengthy title or documentation.

Suppose, at this point, the user wants a report of the current title, but first wants to enter a comment. Hitting FK4 displays and activates windows (12) and (13) as shown in Figure 2b. The user enters his or her comment and then hits FK4 again to print the comment (or FK5 to cancel the printing). Windows (10) and (11) are then redisplayed, and the user is returned to the window that was active before the printing operation. Hitting FK3 causes the document information to be printed. Figure 8 outlines an expanded scenario.

CONCLUSIONS

PANES offers three levels of use: For standard applications, a simple template, similar to that of Figure 7, is easily understood and modified by moderately experienced programmers. For nonstandard applications, new templates can be written, and finally, for very sophisticated applications, all of the parameters and lower level routines of PANES can be accessed.

In the original implementation of PANES, we defined window types as an enumeration of available window types, such as LABEL, USER_ANY, USER_NUMERIC, USER_ALPHA, and MENU. Our original intent was to make window definition easier for the programmer by typing windows in this manner. Instead of windows being defined by a few attributes and their ICTs, the procedures in EDITOR and DISPLAYER "knew" specifically about each predefined window type. We quickly found, however, that there were a large number of distinct window types with many variations, and that each new type required new definition parameters and management routines. The more recent ICT approach has proved to be more succinct and robust. By defining windows in this fashion, programmers have the flexibility to invent "renegade" window types that might not otherwise be allowed. So that PANES did not lose the simplicity of the original typing approach, the interactive SCREEN_DEFINER was created to allow windows to be defined by choice from a menu of commonly used window types. This acts as a buffer between programmers and the details of window definition.

PANES has proved to be surprisingly easy to learn and use. Given the support tools such as SCREEN_DEFINER, the application template, and the options of three levels of use, PANES has been used effectively by programmers with little training and only a few months of programming experience. It has also proved useful in implementing fairly sophisticated applications in various areas. PANES has been used to implement user interfaces for applications in the areas of office automation, knowledge-based (expert) systems, database and document-base management systems, as well as run-of-the-mill data processing. PANES joins the concepts of window-oriented and forms-oriented management to provide a flexible system with which programmers can readily implement user interfaces in a variety of application domains.

REFERENCES

1. Arciszewski, H., and Van Gasteren, E. P/CL: A flexible input processor. *Softw. Pract. Exper. 14*, 12 (Dec. 1984), 1141–1148.
2. Bass, L.J. A generalized user interface for applications programs (II). *Commun. ACM 28*, 6 (June 1985), 617–627.
3. Digital Equipment Corporation. Introduction to VAX-11 FMS. Rep. AA-L318A-TE. Digital Equipment Corporation, Maynard, Mass., Jan. 1983.
4. Draper, S., and Norman, D. Software engineering for user interfaces. In *Proceedings of the 7th International Conference on Software Engineering* (Orlando, Fla., Mar. 26–29, 1984). IEEE Computer Society, Los Angeles, Calif., pp. 214–220.
5. Goldberg, A. *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley, Reading, Mass., 1984.
6. IBM. CICS/VS version I systems/application design guide. SH20-9025, IBM. Available through IBM branch offices.
7. Jacky, J.P., and Kalet, I.J. A general purpose data entry program. *Commun. ACM 26*, 6 (June 1983), 409–417.
8. Konsynski, B., Kottemann, J., Nunamaker, J., and Stott J. PLEXSYS-84; An integrated development environment for information systems. *J. Manage. Inf. Syst. 1*, 3 (Winter 1984–1985), 64–104.
9. Rose, C., and Hacker, B. *Inside MacIntosh*. Vol. 1. Addison-Wesley, Reading, Mass., 1985.
10. Rowe, L., and Shoens, K. Programming language constructs for screen definition. *IEEE Trans. Softw. Eng. SE-9*, 1 (Jan. 1983), 31–39.
11. Wartik, S.P. and Penedo, M. Fillin: A reusable tool for form-oriented software. *IEEE Softw. 3*, 2 (Mar. 1986), 61–68.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; H.1.2 [**Models and Principles**]: User/Machine Systems

General Terms: Human Factors

Additional Key Words and Phrases: Display system, screen management, user dialogue

Received 12/86; revised 7/87; accepted 5/87

Authors' Present Address: Jack W. Stott and Jeffrey E. Kottemann, Dept. of Decision Sciences, University of Hawaii, 2404 Maile Way, Honolulu, HI 96822.