An incidence-matrix-driven panel system for the IBM PC

by P. Halpern
S. M. Roberts
L. Lopez

A set of programs called TRYLON is discussed which permits the application developer to design and create a set of intelligent full-screen panels. These panels serve as a user interface for application programs. The panels and the linkages between them are uncoupled from the application code, thereby reducing programming effort and development time. An incidence matrix is used to describe the graph of a network composed of the panels. Because the paths among the panels are specified by entries in the incidence matrix at panel creation time, there is no need to program the logical constructs for the network.

The introduction of the personal computer has seen an explosion of software development that has placed complex application programs in the hands of both the first-time user and the experienced user. To obtain the maximum benefit from these programs, the user must have a detailed understanding of how to use specific options. Thus, the program developer has the burden of making his application easy to use and to understand. Prior to the advent of the personal computer, the program developer could anticipate the level of expertise that a user of his program would possess. The user was often an expert in a particular discipline or the user was the programmer himself. For these users the interface could be primitive and was often added as an afterthought. Software designed for the personal computer is often directed at a less experienced user, so an additional dimension is added to the program development effort: the need for an effective user interface. The program developer must devote serious time and energy to designing and implementing an intelligent user front end for his program. Such an effort requires a significant increase in the amount of code to be written and logically integrated into the application code. A common user interface is the full-screen panel. (A panel is a recallable full screen of information containing text and fields for input and output of data.) In many applications these are written in the language of the application or in one callable from the application. The amount of code needed to display, to logically link the panels, and to support a keyboard handler can be equal to or greater than the amount of application code generated.

This paper presents a novel approach to recasting the traditional relationship between the function specification in the application code and the user interface, thereby reducing the programming effort and development time for application creation. In this formulation, the panels and the relationships among them are uncoupled from the application code. The panels and their logical connections are considered to be a network in which the nodes are the panels and the connections are the paths. The functional relationship between nodes and paths resides not in the main line of the application program but external to it. The repository for this logical structure is an incidence matrix. The user sees the nodes and paths as full-screen panels and the linkages between panels. At panel definition time, the panels

are linked in a logical manner to form the network. During execution, the end user supplies the required input, requests help, and receives output through the panels. The network is traversed on the basis of decisions captured by the panels and controlled by entries in the incidence matrix.

Early dialog management systems such as ISPF[1] were based on a conversational monitor system[2] and were mainframe-based programs. These programs provided screen generation as part of an overall application development programming environment.

To meet the need to create full-screen panels for use with application programs, several PC-based panel systems have been developed.

To meet the need to create full-screen panels for use with application programs, several PC-based panel systems have been developed. These systems permit the program developer to write text which is used to create "help" panels. Such panels can be invoked by the end user at execution time to obtain on-line help. Furthermore, available systems permit the creation of panels into which an end user can either enter data or display results from the application program. Panels of this type act as input and output mechanisms for the application programs.

Existing PC systems such as EZ-VU[3] help the programmer with panel design by supplying an editor. Although the editors differ in the function they provide, they have many features in common. They usually permit the programmer to specify text as well as input and output data fields that can be arranged into an overall panel design. As part of the data field definitions, the user can select a number of field attributes—for example, field name, data type, maximum and minimum values, output only. At execution time, range checking is carried out for input data. Functions for drawing lines and boxes are frequently provided by the editor.

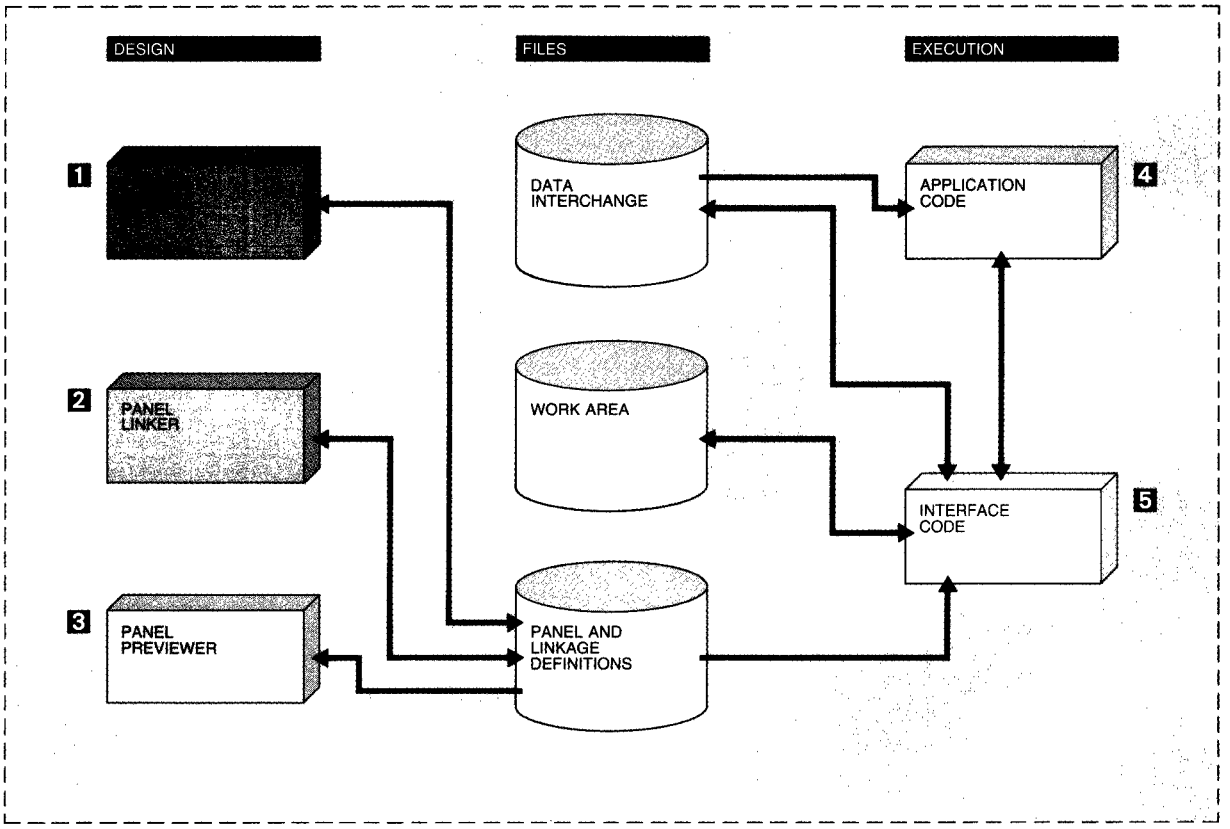
The panel systems usually have several high-level language interfaces. These enable applications written in FORTRAN or Pascal, for example, to access the panels created and to capture data entered into the panels for use by the application program. Similarly, these interfaces transfer output data from the application to panels. These systems take the orthodox approach of accessing the panels serially from within the application program. They do not view the panels as part of a network. As a consequence, although current panel systems have relieved the programmer of the burden of writing code to create and design the panels, the programmer still must write control flow or linkages among the panels and must integrate these into the application code.

TRYLON overview

Figure 1 is a schematic representation of the components of the experimental panel system called TRYLON. TRYLON uses an incidence matrix as the logic repository for panels created and connected in a network. The panels are designed using a full-screen panel editor (Block 1). In addition to the standard data field definitions and other ancillary attributes, the editor permits the definition of branch fields. These are used by the panel linker in defining paths to other panels (Block 2). Associated with each application is a set of files which contains the textual material of the panels, the data field and branch field definitions, and the incidence matrix. In our work, the rows of the incidence matrix may be thought of as destination panels and the columns as source panels. To be more specific, if the source panel includes a menu of options or items, each menu option for that source panel has its own column. Files are used for data interchange between the panels and the application, and for a work area. The panel linker creates the incidence matrix in a file which is accessed at execution time. By using the panel previewer (Block 3), the linked panels can be viewed for desired panel sequencing.

The application program (Block 4) uses the appropriate interface code (Block 5) to access the panels and perform data input and output. The interface code also captures branch field selections and sequences the panels in the order determined by entries in the incidence matrix that was created during the panel-linker sessions. Input and output data transfer between the application program and the interface

Figure 1 Schematic representation of TRYLON



code is accomplished by using a data interchange field or by directly referencing a field on a displayed panel.

An example

The following example illustrates how TRYLON processes panels with the aid of an application program.

Figures 2A through 2E show a series of panels created as the user interface for making two simple calculations involving trigonometry of a right triangle. The extents (lengths) of the input and output areas are designated by the "#" sign. The selection of branches to other panels or exits is specified by the "@" sign. Both signs are special characters which the interface code recognizes and can distinguish from normal text.

Let us briefly examine the sample problem TEXAMP3. Panel 1 contains text announcing a sine and a hypotenuse calculation. Panel 2 accepts user inputs for five angles whose sines are to be computed. Panel 3 is an output panel that lists the sines of the five angles. In addition, Panel 3 contains a menu with three branching options: (1) to redo the sine calculation; (2) to do a hypotenuse calculation; and (3) to exit to DOS through the application program. If the first option is selected, a branch to Panel 2 is executed. If the second option is chosen, the input panel, Panel 4, is displayed and requests that the sides of a right triangle be entered. Panel 5, which presents the results of the hypotenuse calculation, is then displayed.

Figure 2G is a panel summary showing the details of the data fields and the number of branches on a panel. A field name, field type, input or output type, and maximum and minimum field values, as well as a default value, may be specified for each field defined on a panel. For example, during the creation of Panel 4, the first data field was given the name

Figure 2 (A, B, C, D, E) A series of panels created as the user interface for making two simple calculations involving trigonometry of a right triangle; (F panel linkage summary for sine and hypotenuse calculation

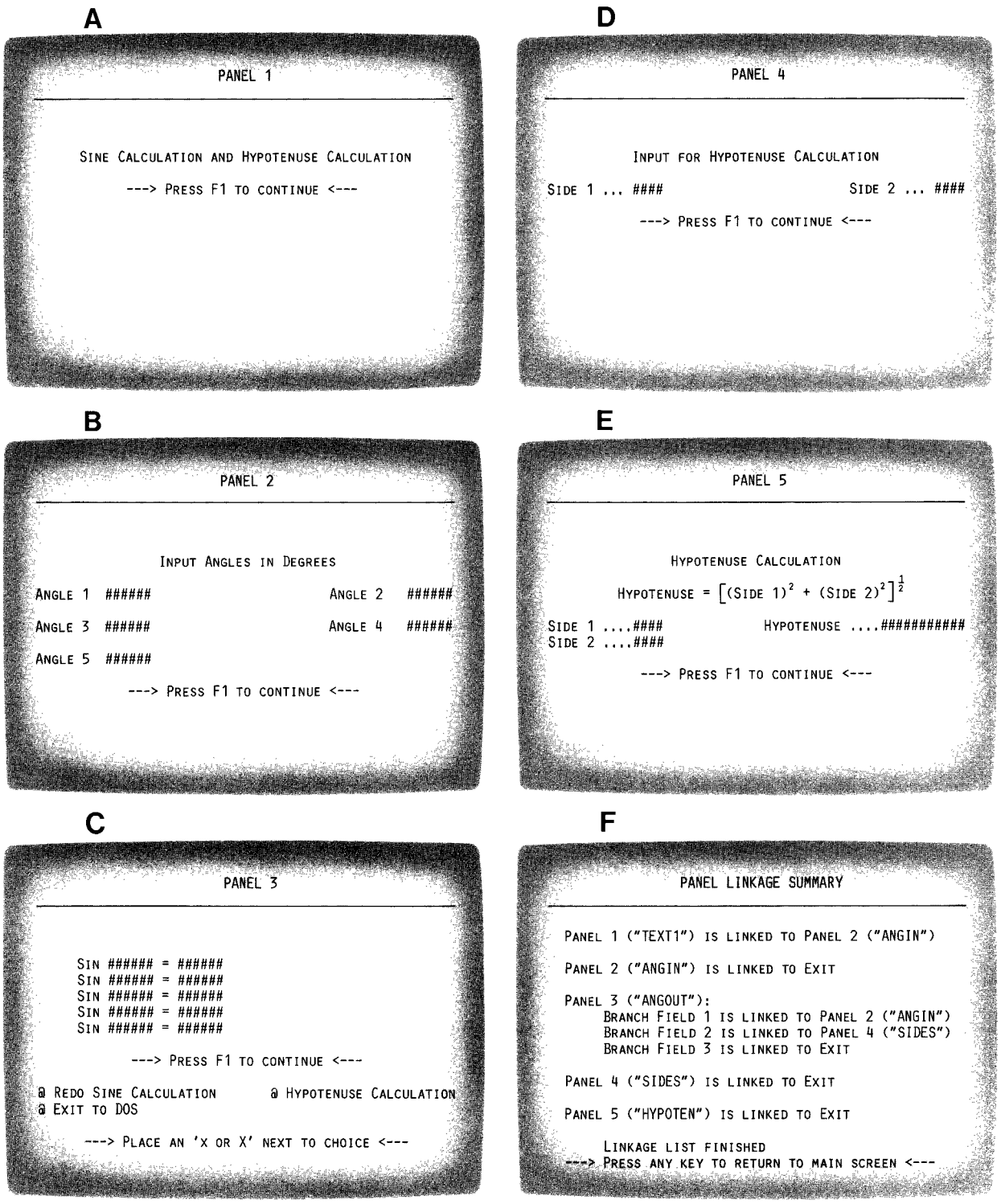
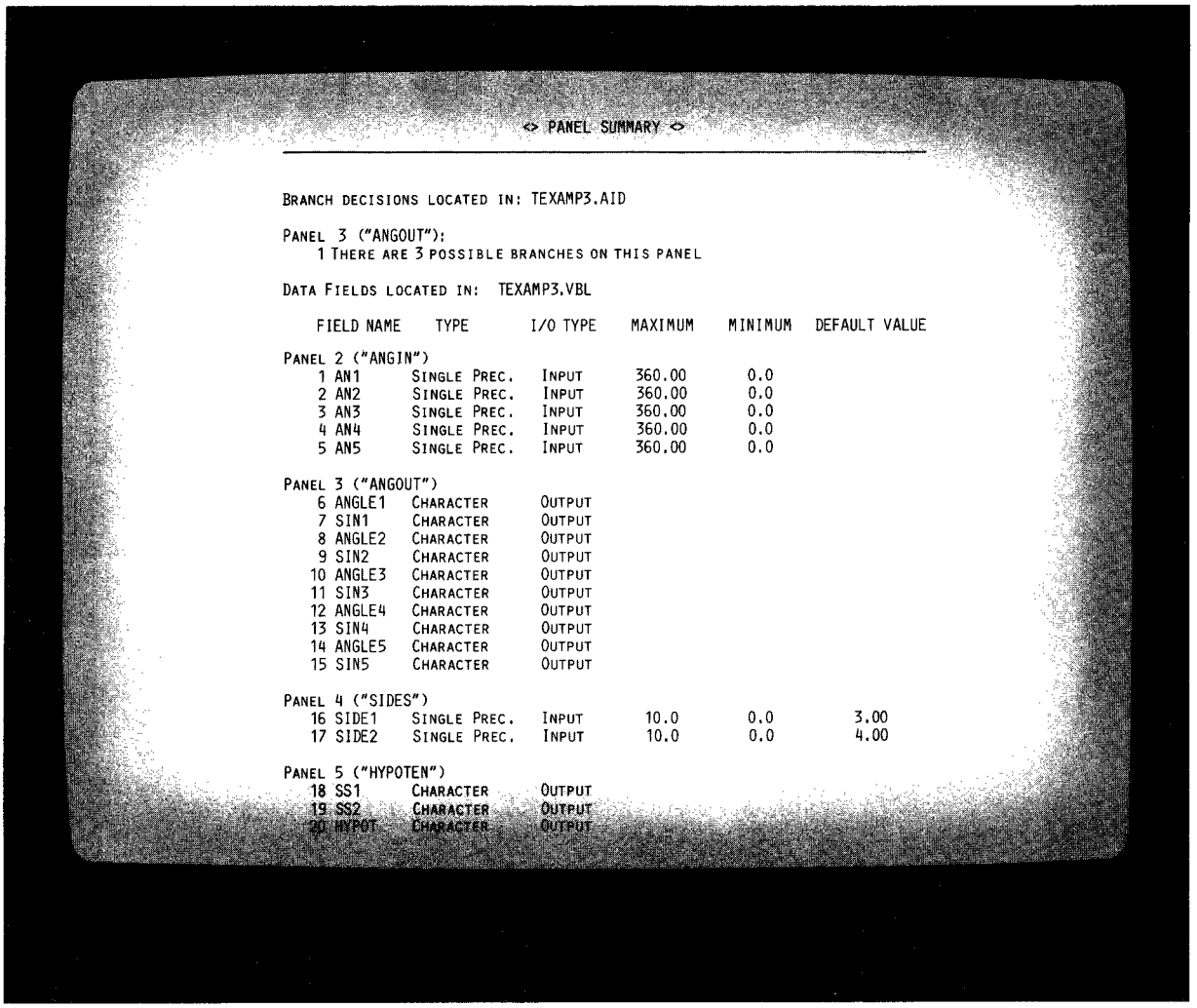


Figure 2 (G) Panel summary for sine and hypotenuse calculation



```
                              <> PANEL SUMMARY <>

BRANCH DECISIONS LOCATED IN: TEXAMP3.AID

PANEL 3 ("ANGOUT"):
    1 THERE ARE 3 POSSIBLE BRANCHES ON THIS PANEL

DATA FIELDS LOCATED IN:  TEXAMP3.VBL

    FIELD NAME    TYPE          I/O TYPE    MAXIMUM    MINIMUM   DEFAULT VALUE

PANEL 2 ("ANGIN")
    1 AN1       SINGLE PREC.   INPUT       360.00       0.0
    2 AN2       SINGLE PREC.   INPUT       360.00       0.0
    3 AN3       SINGLE PREC.   INPUT       360.00       0.0
    4 AN4       SINGLE PREC.   INPUT       360.00       0.0
    5 AN5       SINGLE PREC.   INPUT       360.00       0.0

PANEL 3 ("ANGOUT")
    6 ANGLE1    CHARACTER      OUTPUT
    7 SIN1      CHARACTER      OUTPUT
    8 ANGLE2    CHARACTER      OUTPUT
    9 SIN2      CHARACTER      OUTPUT
   10 ANGLE3    CHARACTER      OUTPUT
   11 SIN3      CHARACTER      OUTPUT
   12 ANGLE4    CHARACTER      OUTPUT
   13 SIN4      CHARACTER      OUTPUT
   14 ANGLE5    CHARACTER      OUTPUT
   15 SIN5      CHARACTER      OUTPUT

PANEL 4 ("SIDES")
   16 SIDE1     SINGLE PREC.   INPUT        10.0        0.0         3.00
   17 SIDE2     SINGLE PREC.   INPUT        10.0        0.0         4.00

PANEL 5 ("HYPOTEN")
   18 SS1       CHARACTER      OUTPUT
   19 SS2       CHARACTER      OUTPUT
   20 HYPOT     CHARACTER      OUTPUT
```

SIDE1. It was defined as an input field for a single-precision number. The allowable range of values was 0.0 to 10.0. During execution, user input values were checked for numerical values falling outside the specified range. A default value of 3.00 would be displayed in that field. Note that all the fields in Panel 3 are defined as output for I/O type, and as character for data type.

The panel linkages for the set of five panels are shown in Figure 2F. The information in Figures 2F and 2G is produced by TRYLON using the editor and linker referred to in Figure 1. Figure 3 shows the traditional graph of the panel linkages (see Figure 2F) and its relationship to the application program. The application requires panel input or output at four locations in the graph. These accesses are indicated by the arrows from the application program to the graph. The arrows mean that the application program invokes the interface program, which displays the indicated panel. The first access is the initial access to Panel 1, which serves as an introduction to the user. A second access follows the sine calculation. The third access follows selection of the hypotenuse calculation, and the final access is after the hypotenuse calculation. Each of the EXIT blocks denotes a return to the application program. The user can choose the EXIT from BR3 of Panel 3, but the

Figure 3 Flowchart of sine and hypotenuse calculation

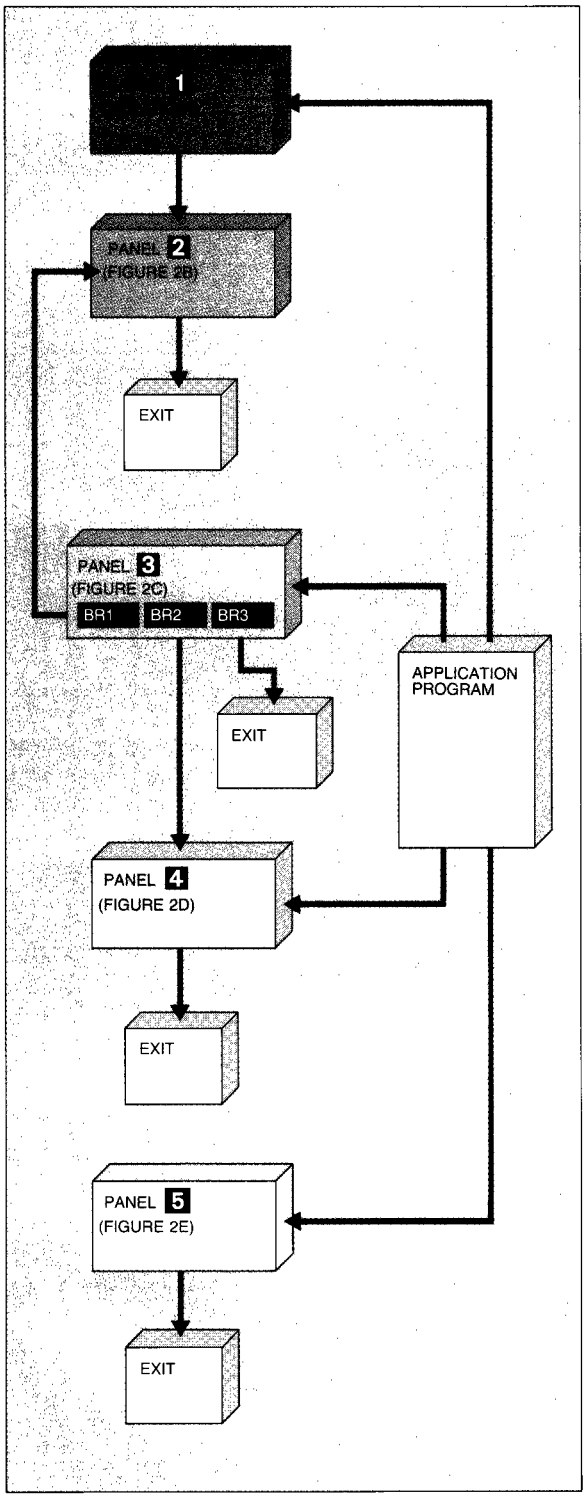
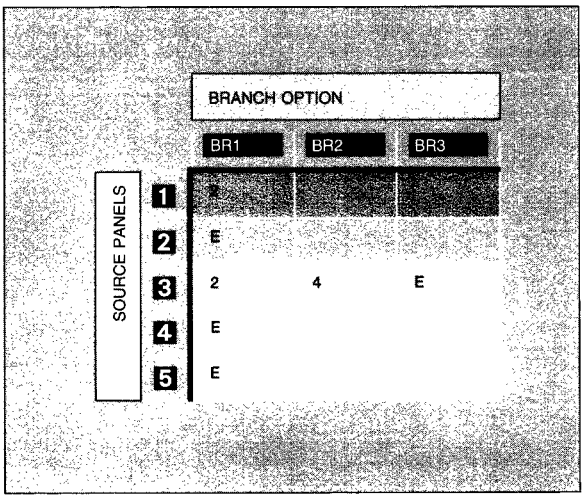


Figure 4 Compact form of incidence matrix for sine and hypotenuse calculation



interface code determines the EXIT from Panels 2, 4, and 5.

The compact incidence matrix for the graph shown in Figure 3 is depicted in Figure 4. The panels which exit have an E in the cell defined by the source panel (row) and the branch number (column). For those panels which do not exit, or do not have branches, a destination panel is specified in the first column (see Panel 1).

The entering of data into a large incidence matrix can be a time-consuming and error-prone chore. The alignment of the entries in the correct rows and columns requires visual, graphic, or text aids. TRYLON eliminates these problems by providing a linker which permits the user to link panels in context. When a source panel is specified, the panel linker displays the panel to be linked (source panel), and the program developer specifies the destination panel for each branching option. The destination panel can be displayed before linking to verify whether the correct link is being specified. Links can be made between any two panels regardless of their positions in the network. Neither the end user nor the program developer ever sees the incidence matrix in any form because the system internally creates the compact form of the incidence matrix from the data supplied. To be specific, consider Figure 5, which shows Panel 3 being displayed by the TRYLON linker program. The destination panel for the branch "Redo Sine Calculation" is set to be Panel 2. The message at the

Figure 5 Creating panel linkages using TRYLON

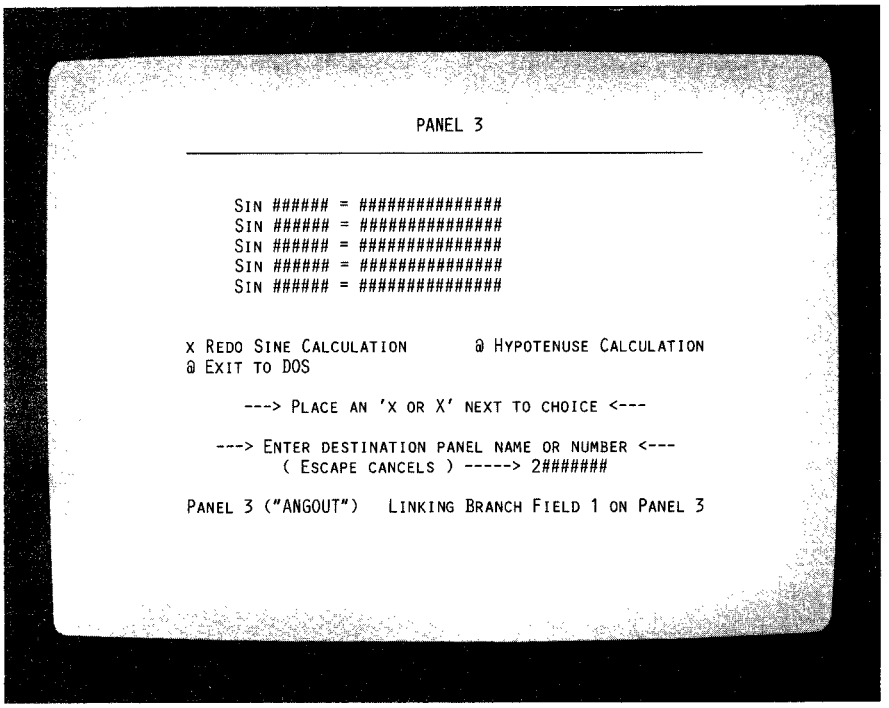
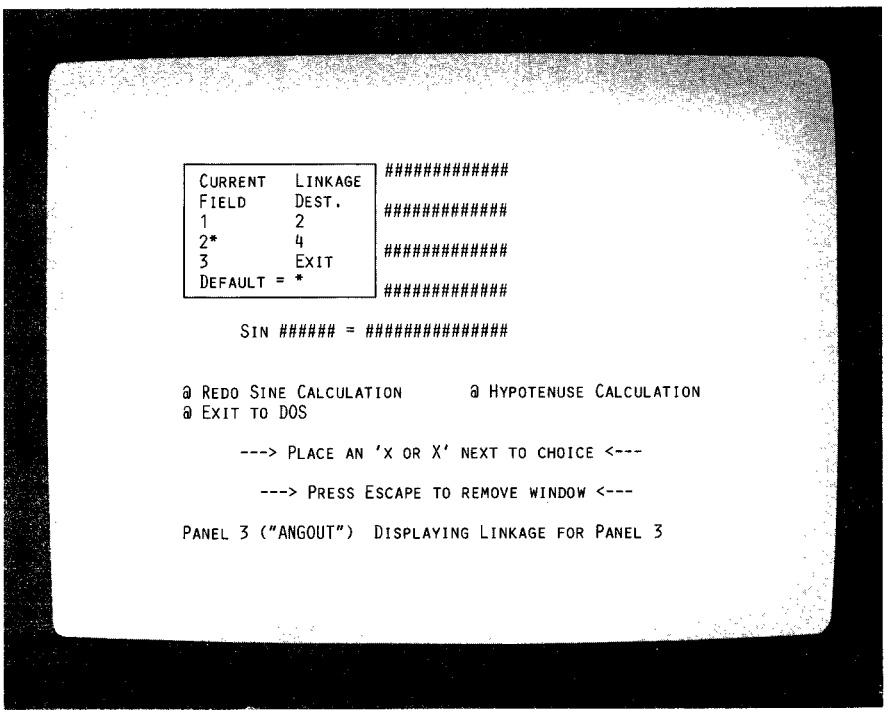


Figure 6 Displaying panel linkages using TRYLON



bottom of the screen gives the current branch field being linked. By pressing the "Enter" key the program developer moves to the next branch field and enters its destination panel. The current linkages for any panel can be reviewed as shown in the pop-up window in Figure 6. Here all the branches and their destination panels have been defined. A choice for the default positioning of the cursor at execution time has also been made by placing an "x" in the desired branch field.

A function for which these panels were designed is the carrying out of two calculations in tandem. That is to say, the problem path consists of the following steps: (1) entering the input data values for the angles, (2) computing the sines, and (3) displaying the results of the sine calculation. Next, the input data for the hypotenuse calculation are entered. Another function consists of running the sine calculation more than once and then exiting without executing the hypotenuse calculation. Other possibilities include running the sine calculation multiple times before executing the hypotenuse computation. These various scenarios are made possible by the menu options in Panel 3.

The TRYLON editor is a full-screen editor capable of accepting both character and numerical data.

Editor

The TRYLON editor is a full-screen editor capable of accepting both character and numerical data. The panels displayed in Figures 2A to 2E were created by the editor prior to running the sample problem. The user has at his disposal a number of convenient editing functions, such as (1) left and right as well as up and down scrolling of the screen, (2) creation of horizontal, vertical, and perpendicular lines, (3) generation of boxes with single or double lines, (4) use of the available ASCII characters as fill patterns for screen areas and boxes, (5) creation of pop-up panels, (6) display of panels with foreground and background colors, (7) inserting, moving, deleting, copying, joining, and splitting lines, and (8) standard keyboard functions such as insert, delete, tab, page up, and page down.

The editor also has global functions for copying, deleting, inserting, and renaming panels within the same application.

In addition, with the editor the user may define data fields and branch fields on the screen within lines 1–23 and as much as 80 characters in width. Lines 24–25 are reserved for messages and instructions. The data fields are identified by a "pound" sign (#) for each character in the field. The branch fields are identified by an "at" sign (@). A panel can have a maximum of 50 data fields and 50 branch fields. The user can define or alter the attributes of a field by pressing Function Key 5 (F5). A window will then pop up adjacent to the field in question, so that the field is visible. The input areas of the window require the user to supply such attribute data as field name (associated with a variable in the application code), field type (character, two- and four-byte integer, and single- and double-precision floating-point), field mode (input, output, or both). A fourth entry in the pop-up window permits the user the option of specifying more attributes, such as default value and maximum and minimum values for the field. At execution time, the default value is displayed in the data field and data entered must conform to data type and mode defined for the field. As a check on the data fields, the user may press Function Key 7 (F7), which displays all the current attributes of the data fields.

For the branch fields the principal attribute set by the user is the default position of the cursor, that is, the initial location of the cursor when the panel is first displayed by the application program. After a panel is designed and its attributes determined, the user saves the panel by pressing the ESC key.

Upon completion of an editing session, the editor produces a panel summary (Figure 2G) and creates three files for each panel. The *filename* for each file is always the application name. The .PAN file contains the panel text, panel names, and panel colors for this application. The .PDF file contains the attribute information for all the data fields on the panel, for example, the total number of data fields on the panel and the width of each field. The information for the branch fields, such as the total number of branch

fields on the panel and the location of these fields on the panel, is stored in the .PCM file. All of these files

It is understood that each language has its particular constructs for dealing with its interface code and TRYLON files.

are used by the interface code at execution time to display the requested panel.

Interface code

To illustrate the interface code, we will use the previous example as a vehicle. Rather than limit the discussion to a language-specific representation of the sine and hypotenuse application, we present a discussion of the application and the interface code using generic terms. It is understood that each language has its particular constructs for dealing with its interface code and TRYLON files. Thus, the interface code written in FORTRAN will have different constructs from that written in BASIC. However, it is informative to describe the data flow to and from panels and the interaction of the incidence matrix for sequencing the panels.

An initialization call to the interface code from the application code is required. The term *call* refers to a generic invocation of the interface code; it does not take on a specific meaning associated with a particular application language. This first call performs a number of housekeeping functions such as defining file numbers and opening files containing the panels and their attributes. A second call is issued to display the first panel. The interface code reads the panel file and ancillary files and displays Panel 1. Control now resides with the interface code. Since Panel 1 has text only, the user presses the F1 key to display the destination panel for Panel 1, namely, Panel 2. The F1 key has been defined as the execution key by passing it as a parameter in the call to the interface code. Panel 2 contains input data fields. The interface code now is in a pause state waiting for the user to enter the angle data. When the user enters the data, the interface code checks for valid number formation and determines whether the data are within the specified range for the data fields. When the user presses F1, the entered angle values are read from the panel. This is achieved by using the information on the ancillary files created by the editor. The x, y location of each data field, the data field extent, type of field (for example, integer or floating-point), and input or output type are available to the interface code. A field name on a panel and its variable name in the application can be associated by parameters passed in calls to access field data. Data read from a large number of input data fields can also be stored in a transfer file to be read by the application code using standard file I/O. After control has been passed to the application program, the data can be read from the transfer file. In either case, Panel 2 has as its destination panel an exit to the application program. Using the supplied angle data, the application program computes the sines.

A call to the interface code sets the next panel number to be displayed as Panel 3, and the values of the sines are written to the screen using the same information as that employed in capturing the data. The format of the data can also be specified.

Panel 3 contains both branch fields and data output fields. The user now chooses one of three available options. The interface code will use the incidence matrix to branch to the destination panel for the option chosen. The decision to redo the calculation will result in Panel 2 being displayed, since it is the destination panel for that choice. When the hypotenuse calculation option is selected, Panel 4 is displayed. Panel 4 contains data input fields for entering information about the length of two sides of a right triangle. The procedure followed is the same as that described for Panel 2. Upon completion of the data input, control is passed to the destination panel for Panel 4, which is an exit to the application. The application reads the values of the sides of the triangle from the transfer file and then computes the hypotenuse. A final call to the interface code specifies that Panel 5 should be displayed, with the value of the hypotenuse and the two other sides of the triangle being displayed in the output fields. Since the destination of Panel 5 is an exit, control returns to the application code and an exit to DOS is executed.

Returning to our discussion of Panel 3, when the *Exit to DOS* option is selected, its destination panel is an exit to the application code. Once in the appli-

Figure 7 (A) Simple tree graph; (B) vertex-edge incidence matrix

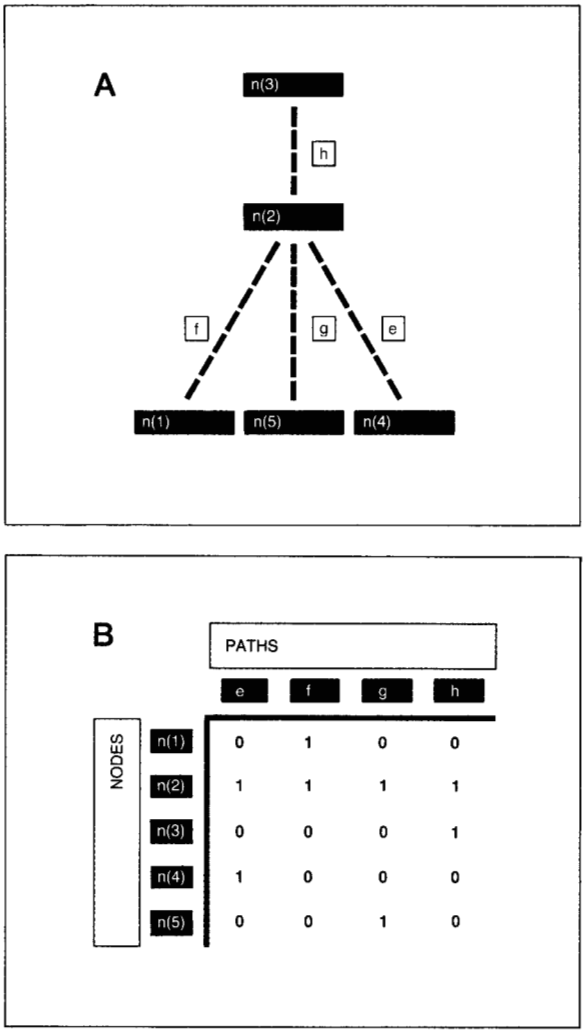
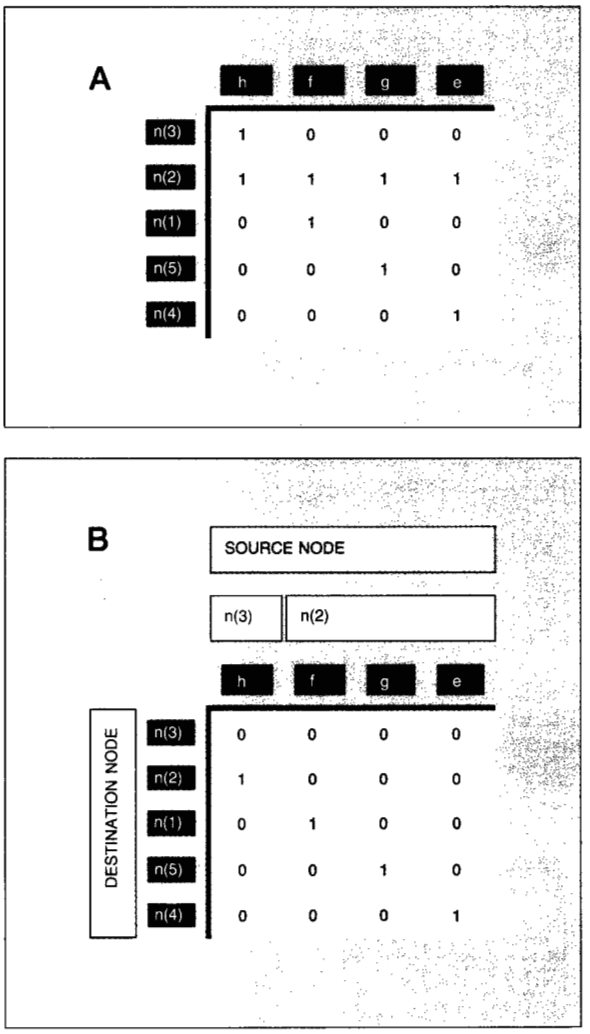


Figure 8 Simple tree graph: (A) Revised incidence matrix; (B) source-node, destination-node incidence matrix



cation, control will be returned from the program to DOS.

Incidence-matrix representation of graphs

TRYLON uses the incidence matrix in a novel method for linking panels. The incidence matrix can be derived by considering that a graph of a network may be represented by a matrix.[4] If we consider a graph with n vertices (or nodes) and p edges (or paths connecting the nodes), and with no self loops, we may define its matrix representation through the $n \times p$ matrix **A**. The n rows of **A** correspond to the n vertices (nodes) and the p columns correspond to the p edges (paths). The matrix **A** with the elements a_{ij} is defined as

$a_{ij} = 1$, if the jth edge (path) is incident on the ith vertex (that is, the path j is connected to the ith node);

$a_{ij} = 0$, otherwise.

The matrix **A** is called the vertex-edge incidence matrix. It is a binary matrix because it contains only the elements 1 or 0.

Figure 7A shows a graph and its vertex-edge incidence matrix. In the figure the vertices (nodes) are labeled n(1), n(2), $\cdots$, n(5) and the edges (paths) are labeled e, f, g, h. Upon adopting the definition of the vertex-edge incidence matrix, we may represent the graph by the incidence matrix in Figure 7B. Here we see, for example, that Path e in Column 1 has a 1 in Row 2 and Row 4, indicating that Path e joins Nodes 2 and 4. Inspection of the incidence matrix in Figure 7B shows that it portrays faithfully the information in the graph in Figure 7A.

The matrix representation in Figure 7B is not the only way to describe the graph in Figure 7A. If we are willing to consider Figure 7A as a directed graph,

The network may be expressed as a graph in which panels are nodes and paths are linkages between panels.

that is, a graph in which the paths are oriented in a specified direction, other interesting representations can be written.

Consider, for example, Figure 7A as a directed graph when the paths are directed from Root n(3) to n(2), and then from n(2) to n(1), n(2) to n(5), and n(2) to n(4). The incidence matrix may be revised to represent this, as shown in Figure 8A.

Another useful representation for the directed graph described in Figure 8A is the incidence matrix in Figure 8B. Here we think of the beginning of a path as a source node and the end of a path as a destination node. For example, in Path h, n(3) is the source node and n(2) is the destination node. An examination of Figure 8B shows that it represents the directed graph as faithfully as Figure 8A. Relative to source and destination nodes, Figure 8B is a particularly useful representation of a directed graph, a form that will be treated more fully in the discussion of the sequencing of panels.

An incidence matrix for panel sequencing

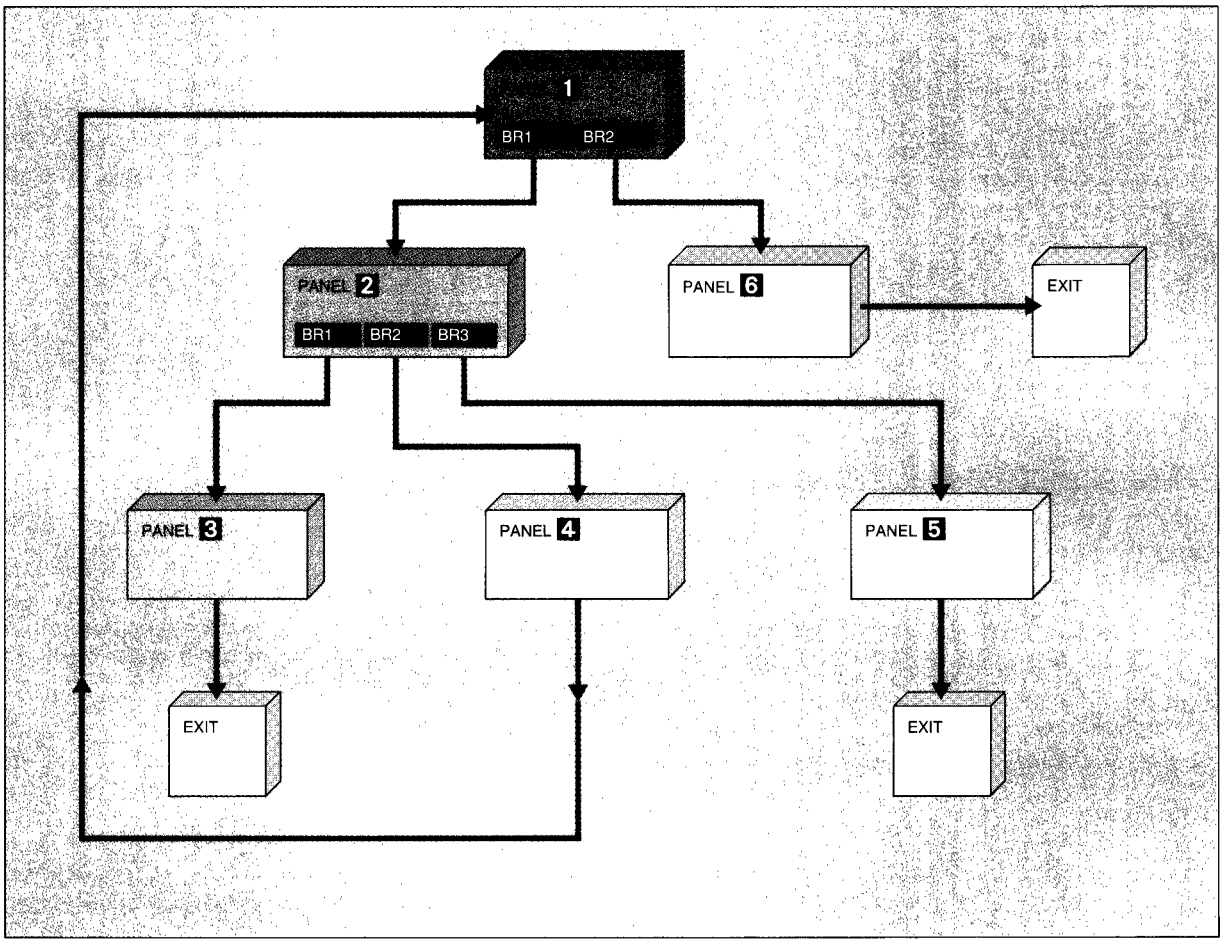
The network through which panels are to be sequenced may be expressed as a graph in which panels are nodes and paths are linkages between panels. The display of one panel followed by another may be thought of as having a source node/destination node relationship. When treated appropriately, such a graph may be processed without writing or rewriting code whenever the graph is altered or traversed. The alteration may include modifying various paths through the graph or adding or deleting nodes in the graph.

Our concern is with the display of panels in a sequence determined by the programmer. Each panel (with the possible exception of the initial root panel) may be considered to be both a source panel and a destination panel. The source panel is the current panel, and the destination panel is the one to which the source panel branches or connects (if there are no branch options). If a panel contains a menu, it may be thought of as Panel P with Branch Option 1 (first menu choice), Panel P with Branch Option 2 (second menu choice), and so on. In other words, a panel with a menu behaves as if it were multiple source panels.

To strengthen our understanding of these concepts, let us consider a graph of panels in Figure 9. Panel 1, the initial panel, contains a menu with two options, labeled BR1 (Branch 1) and BR2 (Branch 2). Panel 1 BR1 is the source panel for the destination panel, Panel 2, while Panel 1 BR2 is the source panel for the destination panel, Panel 6. Panel 2, as a source panel with a menu of three options, branches to Destination Panels 3, 4, and 5 for Branch Options BR1, BR2, and BR3, respectively. Panel 4 differs from the other panels because it branches back to Panel 1. Observe that Panels 3, 5, and 6 branch to EXIT, which is a special destination panel designed to exit the graph or terminate the processing of the graph. In the code, it represents a return to the program that called for the processing of the incidence matrix.

The graph in Figure 9 may be expressed as an incidence matrix in Figure 10. The rows are labeled as the destination panels; the columns are labeled as source panels with menu options for the various branches. For example, Column 1 represents Source Panel 1 BR1 (Branch Option 1), Column 2 represents Source Panel 1 BR2 (Branch Option 2), and Column 3 represents Source Panel 2 BR1. The number 1

Figure 9 Sample panel network



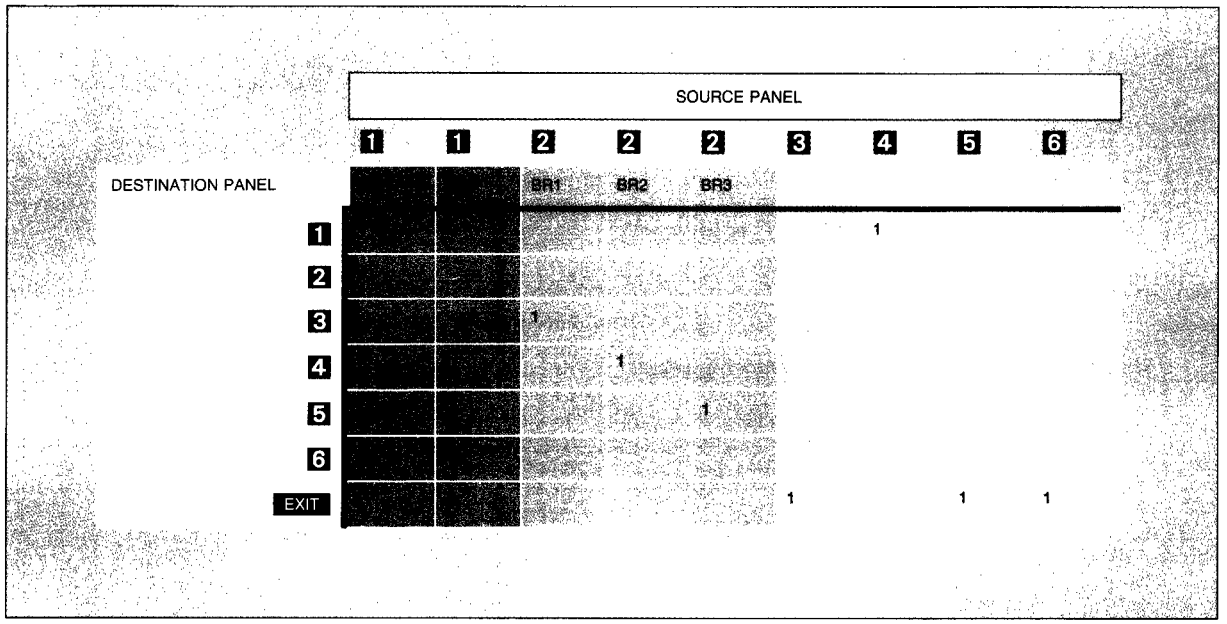
entered in the cell at the intersection of the column labeled Panel 1 BR1 and the row labeled Destination Panel 2 indicates that Source Panel 1, Branch Option 1, connects Source Panel 1 to Destination Panel 2. The entry of a 1 in the cell at the intersection of the column labeled Panel 2 BR2 and the row labeled Destination Panel 4 means that Source Panel 2, Branch Option 2, connects Source Panel 2 to Destination Panel 4. Similarly, one can verify that the incidence matrix in Figure 10 captures the relationships in the graph in Figure 9.

It is interesting that each source panel (with each branch option) connects to one, and only one, destination panel (or EXIT, which may be considered a special destination panel). In the incidence matrix, this translates into one, and only one, entry of 1 in each column. That is to say, the graph is unequivocal.

For substantial systems, the incidence matrix becomes extremely large. If T = the total number of panels excluding the EXIT possibility and b_i = the number of branch options for the ith panel (where $b_i = 1$ for the ith panel that contains no branching option), then the incidence matrix consists of $(T + 1)$ rows and $\sum_{i=1}^{T} b_i$ columns, or a total of $(T + 1) * \sum_{i=1}^{T} b_i$ entries.

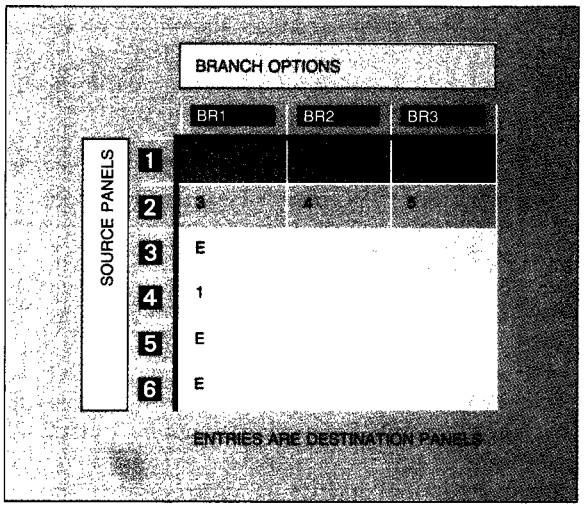
Because the incidence matrix is sparse, the size of the matrix can be considerably reduced by storing only the nonzero entries. We now define a matrix **C** which contains the same information as the incidence matrix but in a compact representation form.

Figure 10 Incidence matrix for sample panel network



In the matrix C, the rows refer to the source panels. That is, Row *i* refers to Source Panel *i*. The columns refer to branch options for each source panel. For example, Column 1 refers to Branch Option 1, and Column 2 refers to Branch Option 2. The entries in the cells contain the destination panel numbers corresponding to the source panels and branch options. To illustrate, let us consider Figure 11, which is the compact representation of the incidence matrix in Figure 10. Figure 11 exhibits, for Source Panel 1 in Row 1, two destination panels, 2 and 6, in sequence as specified by the branching options BR1 and BR2 of Source Panel 1. Source Panel 2 in Row 2 lists in sequence Destination Panels 3, 4, and 5 corresponding to BR1, BR2, and BR3 options of Source Panel 2. In Row 3 for Source Panel 3, the E in Column 1 refers to EXIT as the destination panel. When there are no branching options, as in Source Panels 3, 4, 5, and 6, the destination panel appears in Column 1 (under BR1). The reader can verify that Figure 11 does indeed reproduce the sequence of events specified by Figures 9 and 10.

Figure 11 Compact form of incidence matrix



During execution, the panels are sequenced on the basis of the paths specified in the incidence matrix or its compact representation. To change the sequencing would require altering the entries in the incidence matrix or the compact representation. In actual practice, the compact representation is employed.

The processing of the incidence matrix is completely independent of the entries in the incidence matrix.

Thus the processing code, supplied with the number of rows and columns in the compact form of the incidence matrix, can handle the addition or deletion of panels without the need for logic programming by the program developer.

Conclusion

We have presented a novel way to link panels without the use of a command language or logic programming on the part of the programmer. This linkage was achieved by the introduction and use of the incidence matrix. Since the incidence matrix is created independently of the application program, text panels may be added, deleted, or reordered in any sequence without recompiling the application code. These alterations are possible because the processing of the incidence matrix is independent of the contents of the matrix.

Text panels may be added, deleted, or reordered in any sequence without recompiling the application code.

The incidence matrix can be expanded to enhance a variety of tasks. These include (1) linking to other programs, as well as to panels, (2) linking graphics screen displays, and (3) linking to a host for carrying out computationally intensive functions. In fact, the incidence matrix may be used as an executive that directs the flow of a series of related programs.

We have implemented a running program, TRYLON, consisting of an editor for creating panels and a linker for generating connections among the panels and linking them to the application program. TRYLON is being evaluated and used at IBM internal sites, and, in addition, has been sent to a small number of university researchers for their evaluation and use.

Cited references

1. *Interactive System Productivity Facility, Dialog Management Services*, Program No. 5668-960, IBM Corporation; available through IBM branch offices.
2. *Structured Programming Facility/Conversational Monitor System*, Program No. 6C20-0370, IBM Corporation; available through IBM branch offices.
3. *EZ-VU Development Facility for the IBM Personal Computer*, Product No. 6410980, IBM Corporation; available through IBM branch offices.
4. N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

Paul Halpern *IBM Academic Information Systems, Palo Alto Scientific Center, P.O. Box 10500, Palo Alto, California 94303.* Dr. Halpern has been with IBM since 1968. He has published papers in the fields of numerical weather prediction, air pollution meteorology, terrestrial radiative transfer, solar energy harvesting, real-time data acquisition, and digital image processing. He received a B.S. degree from the City College of New York in 1961, an M.S. from New York University in 1963, and a Ph.D. in atmospheric science from the University of California at Davis in 1975. Dr. Halpern is a member of the American Meteorological Society and the American Geophysical Union. He is currently a staff member at the ACIS, Palo Alto Scientific Center.

Sanford M. Roberts *IBM Academic Information Systems, Palo Alto Scientific Center, P.O. Box 10500, Palo Alto, California 94303.* Dr. Roberts is currently associated with the IBM Corporation's Palo Alto Scientific Center. In 1948 he received a B.S. in petroleum engineering from the University of Pittsburgh, where he held an honor scholarship; he was a Research Fellow at the University of California at Berkeley, and received an M.S. in mechanical engineering from that institution in 1950; his Ph.D. in chemical engineering was awarded by Stanford University in 1953. Dr. Roberts has worked as a research engineer and applied mathematician in the areas of process control, optimization, and numerical analysis; he has published a book on dynamic programming in process control.

Louis Lopez *IBM Academic Information Systems, Palo Alto Scientific Center, P.O. Box 10500, Palo Alto, California 94303.* Dr. Lopez joined IBM at the Houston Scientific Center in 1966, working on the application of computers to engineering and scientific problems. From 1974 to 1979 he was manager of process analysis, first at the Houston Scientific Center and then at the Palo Alto Scientific Center. In 1980 he took a one-year sabbatical to the General Products Division plant in San Jose, returning in 1981 in his current position as a member of the technical staff of the Palo Alto Scientific Center. Dr. Lopez received a B.M.E. in 1961 and an M.S.E. in 1962 from the University of Florida, and a Ph.D. in 1966 from Rice University, all in mechanical engineering.

Reprint Order No. G321-5294.