Graphical modelling using contextually implied functions

C. J. Evangelisti and S. P. Morse*

International Business Machines Corporation, Research Division, Yorktown Heights, New York, USA

Many physical problems can be represented by block diagrams or, more generally, by topological graphs. The nodes or blocks correspond to objects with specified attributes. The edges or lines between nodes represent relations between the objects and may also have attributes.

A modelling (drawing) system is described that allows for block diagrams to be entered into a computer via a graphical console such as the IBM 2250. The techniques involved do not require a user to explicitly tell the machine which drawing function he wants executed; the operands he chooses determine the function. Thus, the system eliminates the need for function keys or light keys while the model is being created and permits the user to operate solely with a light pen. Keys are used only for functions that are secondary to the creation of the model, such as the filing function for example.

A set of functions useful for creating block diagrams is presented. These functions provide the ability to copy blocks, draw lines between blocks, and erase lines or blocks. Functions pertaining to the attributes provide the ability to specify attributes, change attributes, and associate attributes with lines or blocks. Finally, a file structure is described and filing and retrieval functions are presented.

(Received May 1970)

Many problems encountered by engineers and scientists are solved by representing physical systems by abstract models made up of interconnected symbolic elements called *blocks*. The relative positions of the blocks convey no information. Some examples of such models are electrical networks, program flowcharts, and simulation block diagrams.

Such models are usually processed to determine the behaviour of the physical system represented. If the processing is to be done on a digital computer, a description of the model must be given to the computer. A graphical terminal consisting of a lightpen and display screen provides a user with a pencil and paper substitute for creating a model and, at the same time, directly conveys the model to the machine.

Modelling on a graphical terminal usually involves making copies of blocks from some initial set of blocks and drawing lines between the copies. Hence, the terminal must provide the user with the ability to perform functions such as copying blocks and drawing lines. An explicit command is usually given each time such a function is performed. Thus, the user must specify both the function and any operands of the function. For example, when making a copy of a block, he must specify that he wants the copy function (usually by pressing a function key or pointing to a light key) and indicate which block is to be copied and where the copy is to appear. Many examples of such graphical systems can be found in the literature (Sutherland, 1963, 1966, Hornbuckle, 1967, Ellis and Sibley, 1967). This paper describes a method of graphical modelling in which the user is not required to explicitly specify which function is to be executed when he creates his model (he must give explicit commands only for functions that are secondary to the creation of the model, such as the filing function). Instead he selects his operands; the choice of the operands uniquely determines a function. Specifying functions in this manner eliminates the need of function keys or light keys to give explicit commands.

Modelling

Preliminary remarks

Modelling, as used in this paper, shall refer to the process of representing a physical system by a model composed of interconnected blocks (Baskin and Morse, 1968). The screen on a graphical terminal used for modelling is separated into two spaces. One space contains an initial set of blocks and is called the *reference space*. The other space is called the *modelling space* and contains copies of the blocks in the reference space and lines drawn between the copies. In the method of modelling presented here, the two spaces are separated by a horizontal line on the display. The reference space is the region below the horizontal line and the modelling space is the region above the horizontal line.

Several guidelines were followed in designing the set of modelling functions presented in this paper. The functions were to be both easy to learn and easy to use. Ease of learning requires that the rules for executing the various functions be simple, natural, and consistent with each other. Ease of use requires that the user be able to perform his desired task in relatively few steps, be able to stop in time before taking a wrong step, and be able to undo the effects of any wrong steps. In order to satisfy these two principles, it was decided that two separate actions should be required to execute any function. The two actions correspond to selecting two operands which in turn determine the function. Each operand selection requires a confirmation by the user and the selection can be changed prior to giving the confirmation. Feedback in the form of visual output is given to the user before the user gives the confirmation thereby letting the user know in advance what the result would be if he were to confirm his selection.

Modelling functions

A set of functions for modelling at a graphical terminal must, at a minimum, provide the user with the ability to copy blocks from the reference space into the modelling space, draw lines between (copies of) blocks in the modelling space, and erase items (lines or blocks) from the modelling space. In addition, since each function is implicitly called for by selecting two operands, a function that does nothing would be useful in the event that an incorrect first operand is chosen. Such a set of modelling functions is summarised in **Table 1** (the last function in this table will be discussed later).

The components of a topological drawing are blocks, lines and points. Hence the possible operands that can be selected in

***Presently with the Dept.** of Electrical Engineering and Computer Sciences, University of California, Berkley.

Table 1 Set of modelling functions

D	Draw a line between two points in the modelling space
C	Copy an item from the reference space into the modelling space
E	Erase an item in the modelling space
N	Do-nothing
S	Select an item as a first operand for the the next function

order to specify a function consist of blocks, lines and points in either the reference or modelling space. The points are either attacher points of blocks, endpoints of lines, or new points selected by placing the lightpen on a blank part of the screen. The lines are lines drawn between the points but are not the lines making up the pictorial representation of a block. Since the reference space is to serve basically as a reservoir of modelling blocks, it doesn't seem fruitful at this stage to speak of lines and hence endpoints of lines in the reference space. Hence lines and endpoints of lines in the reference space have been ruled out as possible operands. As a further simplification, lines in the modelling space can also be dropped from the set of possible operands since, in most graphical systems, lines in the model are selected by choosing a point (other than the endpoint) of the line. Thus, topologically, the original line could be considered as two lines having this chosen point as a common endpoint. The remaining possible operands, together with their definitions and mnemonics, are listed in **Table 2.** The operands can be partitioned into two sets called the *reference set* and the *modelling set*. The mnemonic for each operand in the reference set begins with an *R* and the mnemonic for each operand in the modelling set begins with an *M*.

The previous discussions on ease of learning and ease of use give justification for a method of specifying functions using two operands with confirmation and feedback. The following is a sequence of events, consistent with that discussion, which must occur before a function is executed. Initially, the user places the lightpen* on the screen thereby starting the selection process for the first operand. Any item now detected by the lightpen becomes the intended first operand and starts flashing. When the user removes the lightpen from the screen, the last detected item becomes the first operand and remains flashing. The user again places the lightpen on the screen thereby starting

Table 2 Possible operands

Rb	A block in the set of blocks in the reference space selected by placing the lightpen on a line of the block
Ra	An attacher point on a block in the set of blocks in the reference space selected by placing the lightpen on the attacher point
Rn	A new point in the reference space selected by placing the lightpen on a blank part of the reference space (a raster scan takes place)
Mb	A copy of a block in the modelling space selected by placing the lightpen on a line of the block
Ma	An attacher point on a copy of a block in the modelling space selected by placing the lightpen on the attacher point
Me	An endpoint of a line in the modelling space providing the endpoint does not lie on an *Ma* (if several lines have a common endpoint not lying on an *Ma*, only one *Me* exists at that point)
Mn	A new point in the modelling space selected by placing the lightpen on a blank part of the modelling space (a raster scan takes place)

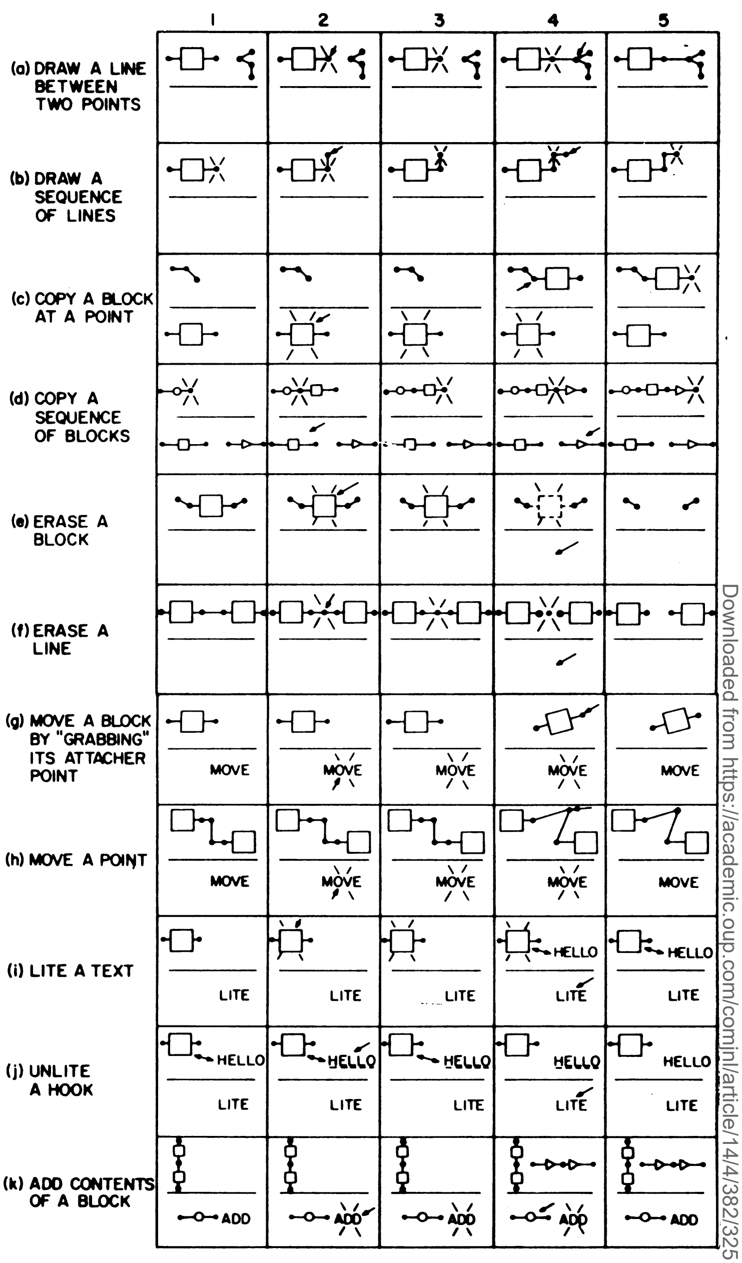
1
2
3
4
5
(a) DRAW A LINE BETWEEN TWO POINTS
(b) DRAW A SEQUENCE OF LINES
(c) COPY A BLOCK AT A POINT
(d) COPY A SEQUENCE OF BLOCKS
(e) ERASE A BLOCK
(f) ERASE A LINE
(g) MOVE A BLOCK BY "GRABBING" ITS ATTACHER POINT
(h) MOVE A POINT
(i) LITE A TEXT
(j) UNLITE A HOOK
(k) ADD CONTENTS OF A BLOCK
MOVE
LITE
HELLO
ADD

Fig. 1. Various steps in execution of functions

1. Screen appearance before function is performed (the horizontal line separates modelling and reference spaces).
2. Operand is being selected (arrow represents lightpen and diagonal lines represent flashing).
3. Selection of operand has been confirmed (function is performed in B and D since these cases started with an implicit operand selected by the previous function).
4. Operand is being selected.
5. Selection of operand is confirmed and function is performed.

the selection process for the second operand. Any item now detected by the lightpen becomes the intended second operand. The result of the function implied by the first and (intended) second operand is temporarily displayed on the screen. When the user removes the lightpen from the screen for the second time, the flashing of the first operand ceases, the last detected item becomes the second operand, and the function implied by these two operands is executed.

The following is a detailed description of the functions as described so far (Table 1) and rules for determining which function is implied by a given pair of operands. The rules of association between functions and operand pairs were chosen so that the resulting functions would be the most obvious functions that a user would expect from a given operand pair

*The ideas presented in this paper are indeed input media independent. However, for ease of description, a lightpen input will be assumed.

and, conversely, the particular operand pair would be the most obvious one that the user would want to select in order to perform a particular function.

1. If both operands are the same item, a DO-NOTHING (N) is executed. Thus, a first operand selected by mistake can be easily cancelled.
2. If both operands are in the reference set, a DO-NOTHING (N) is executed.
3. If both operands are in the modelling set and are not the same item, a DRAW (D) is executed (see **Fig. 1a**). A line is drawn between the selected point (*Ma*, *Me*, or *Mn*) of the first operand (if a block (*Mb*) is the operand, the last attacher point of that block is used by default) to the selected point (*Ma*, *Me*, or *Mn*) of the second operand (if a block (*Mb*) is the operand, the first attacher point of that block is used by default).
4. If one of the operands is in the reference set and is not a new point (*Rn*) and the other operand is in the modelling set, a COPY (C) is executed (see **Fig. 1c**). A copy of the reference block (*Rb*) appears in the modelling space. (If an attacher point (*Ra*) was the operand in the reference set, a copy of the reference block containing *Ra* appears in the modelling space.) If both operands were points (*Ra*, *Ma*, *Me*, or *Mn*), the copy is made such that these points are joined. If a reference block (*Rb*) is one of the operands, its first attacher point† (*Ra*) is used by default; if a model block (*Mb*) is one of the operands, its last attacher point (*Ma*) is used by default.
5. If one of the operands is a new point in the reference set (*Rn*) and the other operand is in the modelling set, an ERASE (E) of the operand in the modelling set is executed (see **Fig. 1e**). An erase of an attacher point (*Ma*) is defined as an erase of all blocks and lines incident on the attacher point. Similarly, an erase of an endpoint is defined as an erase of all lines incident on the endpoint. Note that ERASE is treated as the opposite of COPY. Whereas in a COPY the reference space acts as a 'source' of new items for the modelling space, in an ERASE it acts as a 'sink' for unwanted items.
6. If a line in the modelling set is pointed at by the lightpen, that line is decomposed into two lines at the point where the lightpen hits the line and this point (*Me*) becomes the intended operand (see **Fig. 1f**). It is this rule that allows for the simplification previously mentioned whereby lines are not considered as operands.

A common criticism to such a system is that certain functions would be cumbersome to use if one always had to specify two operands. For example, consider drawing a sequence of consecutive conected lines. It would be desirable to draw such a sequence of lines merely by specifying each successive endpoint. Similarly, consider making successive copies of blocks so that they are connected end to end. It would be desirable to make such copies by merely pointing to the successive reference blocks. These examples would require that one operand of the draw function or the copy function be specified by the previously executed function. Thus, it would be desirable to have a function whose only end product is the selection of a first operand for the succeeding function. This function is the *select* function (S) listed in Table 1. The following two rules relate to the behaviour of this function.

7. All draws (D) that terminate on a new point (*Mn*) are followed by SELECTING (S) that point (which has now become an *Me*) as the first operand for the next function (see **Fig. 1b**).
8. All copies (C) for which a reference block (*Rb*) is one of the operands are followed by SELECTING (S) the last attacher point of the block copied into the modelling space (*Mb*) as the first operand for the next function (see **Fig. 1d**).

The above rules are summarised in **Table 3** and in **Fig. 2.**

Table 3 Relation between operands and functions

	FIRST OPERAND \ SECOND OPERAND	REFERENCE			MODEL			
		Block (*Rb*)	Attacher Point (*Ra*)	New Point (*Rn*)	Block (*Mb*)	Attacher Point (*Ma*)	Endpoint (*Me*)	New Point (*Mn*)
REFERENCE	Block (*Rb*)	Do Nothing (N)	Do Nothing (N)	Do Nothing (N)	Copy (C) Select (S)	Copy (C) Select (S)	Copy (C) Select (S)	Copy (C) Select (S)
	Attacher Point (*Ra*)	Do Nothing (N)	Do Nothing (N)	Do Nothing (N)	Copy (C)	Copy (C)	Copy (C)	Copy (C)
	New Point (*Rn*)	Do Nothing (N)	Do Nothing (N)	Do Nothing (N)	Erase (E)	Erase (E)	Erase (E)	Erase (E)
MODEL	Block (*Mb*)	Copy (C) Select (S)	Copy (C)	Erase (E)	Draw (D)	Draw (D)	Draw (D)	Draw (D) Select (S)
	Attacher Point (*Ma*)	Copy (C) Select (S)	Copy (C)	Erase (E)	Draw (D)	Draw (D)	Draw (D)	Draw (D) Select (S)
	Endpoint (*Me*)	Copy (C) Select (S)	Copy (C)	Erase (E)	Draw (D)	Draw (D)	Draw (D)	Draw (D) Select (S)
	New Point (*Mn*)	Copy (C) Select (S)	Copy (C)	Erase (E)	Draw (D)	Draw (D)	Draw (D)	Draw (D) Select (S)

Note: If both operands are the same item, a DO-NOTHING (N) is executed.

Incorporating attributes into the model

Attributes, represented by alphanumeric text, are usually associated with lines or blocks in a model. For example, values of resistance are associated with each resistor in an electrical network, program statements are associated with blocks in a program flowchart, and transitional probabilities are associated with the lines connecting the states of a Markov Chain. Thus, functions are needed that provide the ability to create and modify text, and relate the text with blocks or lines in the model.

An alphanumeric keyboard can be used to enter text. A string

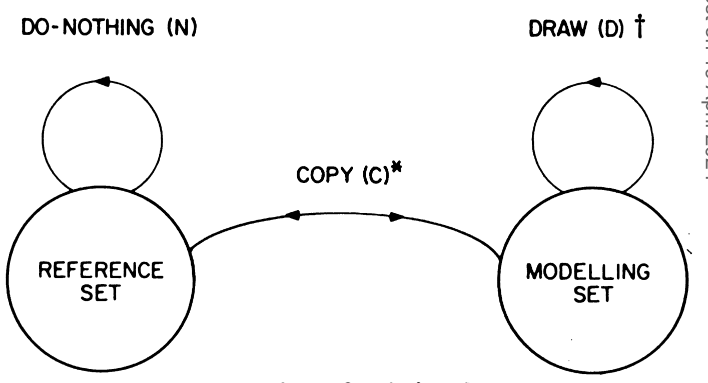

Fig. 2. Graph representation of relation between operands and functions

*If the operand in the reference set is a new point (*Rn*), an erase (E) is executed instead of a copy (C).

†If the two operands are the same, a do-nothing (N) is executed instead of a draw (D).

†It is assumed that all attacher points of a block have been classified as inputs or outputs and that the attacher points have been ordered so that the inputs precede the outputs. Hence a draw or copy function having two blocks as operands will connect an output of one to an input of the other. This is a natural default option.

of characters from the alphanumeric keyboard can be considered as an operand for the modelling functions. While the string is being generated, the string is an intended operand and may be edited as it is being typed; when a special terminal character appears in the string, the intended operand becomes the operand. If a special cancellation character appears in the string, the intended operand is cancelled.

The creation of text involves specifying the desired location of the text and the text itself. Hence, if the first operand is a point in the modelling space and the second operand is a string of characters, text will be created starting at that point. The created text can then be used as an operand for other functions. The created text may be modified by selecting the text as a first operand. The second operand is the modification and is entered from the alphanumeric keyboard. A proposed set of text editing features useful in both the creation and modification functions is found in the Appendix.

A function is required that relates text to a block or a set of connected lines. (Note that a set of connected lines is topologically equivalent to a node.) The text is one operand of this function. The other operand is either a block or an endpoint of a line(s) in the model. (Recall that if a line is pointed at, an endpoint on the line is created and becomes the operand.) The relation can be visually represented by a line with an arrowhead on both ends. Such a line is called a *hook*.

Thus, the consideration of text leads to three additional functions and three additional operands. The functions provide the ability to create text, modify text, and relate text to items in the model. The operands are text in the modelling space, text in the reference space, and alphanumeric character strings. These functions and operands together with their mnemonics are listed in **Table 4.** The modelling set of operands is partitioned into two subsets called the *textual modelling subset* consisting of text and the *graphical modelling subset* consisting of all other items that can appear in the modelling space.

The above discussion of functions necessary for the inclusion of text results in the following additions and modifications to the detailed function descriptions given previously.

Table 4 Functions and operands involved with text

FUNCTIONS	
C	Create text
M	Modify text
R	Relate text to a block or a set of connected lines in the modelling space
OPERANDS	
An	String of alphanumeric characters
Mt	Text in the modelling space
Rt	Text in the reference space

3. If both operands are in the graphical modelling subset and are not the same item, a DRAW (*D*) is executed. (This rule was previously stated to include the entire modelling set).

3*a* If both operands are in the textual modelling subset, a DO-NOTHING (*N*) is executed.

3*b* If one operand is in the graphical modelling subset and the other operand is in the textual modelling subset, a RELATE (*R*) is executed. This function relates the text (*Mt*) corresponding to the operand in the textual modelling subset to the block (*Mb*) or lines corresponding to the operand in the graphical modelling subset (if the operand in the graphical modelling subset is an attacher point (*Ma*) on a block, that block is taken as the operand; if it is an endpoint (*Me*), the set of connected lines containing those lines incident on the endpoint is taken as the operand; if the operand is an attacher point (*Ma*) common to several blocks, the text is related to each of these blocks). Note that text can be related only to blocks or sets of connected lines. One text can be related to several blocks and/or sets of connected lines and a block or set of connected lines can be related to several texts.

4. If one operand is in the reference set and is not a new point (*Rn*) and the other operand is in the modelling set, a COPY (*C*) is executed. A copy of the reference block or reference text appears in the modelling space. If a reference text (*Rt*) is one of the operands, the location one space to the left of its first character is used as the point which lines up with a point in the modelling space. If a modelling text (*Mt*) is one of the operands, the location one space to the right of its last character is used as the point at which a point of the item in the reference space lines up‡. If a reference block (*Rb*) with related text (*Rt*) is one of the operands, the text and the hook are also copied.

5. If one of the operands is a new point in the reference set (*Rn*) and the other operand is in the modelling set, an ERASE (*E*) of the operand in the modelling set is executed. An erase of a block (*Mb*) or a text (*Mt*) implies an erase of all hooks incident on these items.

9. All copies (*C*) of texts are followed by SELECTING (*S*) the text copied into the modelling space (*Mt*) as the first operand for the next function.

The above rules are summarised in **Table 5** and **Fig. 3.**

Supplementary modelling functions

Certain functions are not involved in creating the model; they alter the appearance of the model without changing the meaning. An example of such a function is moving a block while maintaining all connections to that block. These supplementary functions will be characterised by using the names of the functions as one of the operands. Thus, the names of such functions will appear as light keys in a corner of the screen.

Moving items around without breaking any lines changes the appearance of the model but not its meaning. This is desirable for reformatting a cluttered portion of the model. A function that accomplishes this is called the move function. The move function is defined by selecting a light key labelled MOVE as one of the operands. For the purpose of the move function, the model is considered as being made up of rigid bodies (blocks and texts), rubber bands with zero coefficients of elasticity (lines and hooks), and pins (attacher points and endpoints). The actual details of the move were chosen so as to make the moving as simple as (and analogous to) the process of driving an automobile. Hence the move function will provide for pure translation, pure rotation, and any combination of the two. A description of the use of the move function follows.

1. MOVE + *Mb*—as long as the block remains the intended second operand, all pins corresponding to attacher points on the block are removed and the block translates (without rotating) by following the lightpen. Hence, all lines (rubber bands) connected to the attacher points of the block stretch. If any other attacher points are joined to attacher points on the block, a rubber band (line) is connected between these two attacher points. Any text related to the block moves with the block.
2. Move + *Ma*—all pins corresponding to attacher points of the block containing the selected attacher point are removed and a ball bearing is placed at the centroid§ of

‡This somewhat arbitrary choice of alignment was chosen so that two texts can be concatenated.

§Centroid can be defined when a symbol for a block is created (during a separate phase). Otherwise the centroid can be taken to be the centre of the area of the symbol for the block if the symbol is a closed figure; if the symbol is not a closed figure, the centroid can be taken to be the centre of mass of the attacher points of the block.

Table 5 **Relation between operands and functions with text considered**

	SECOND OPERAND / FIRST OPERAND	REFERENCE: Text (*Rt*), Block (*Rb*), Attacher Point (*Ra*), New Point (*Rn*)	MODEL: Text (*Mt*), Block (*Mb*), Attacher Point (*Ma*), Endpoint (*Me*), New Point (*Mn*)	Keyboard (*An*)
REFERENCE	Text (*Rt*) Block (*Rb*) Attacher Point (*Ra*) New Point (*Rn*)	Do Nothing (*N*)	Copy (*C*) Select (*S*) Copy (*C*) Erase (*E*)	Do Nothing (*N*)
MODEL	Text (*Mt*) Block (*Mb*) Attacher Point (*Ma*) Endpoint (*Me*) New Point (*Mn*)	Copy (*C*) Select (*S*) | Copy (*C*) | Erase (*E*) Do-Nothing	(*N*) Relate (*R*) | Draw (*D*) | Draw (*D*) Select (*S*) Modify Text	(*M*) Create Text (*T*)

Note: If both operands are the same item, a DO-NOTHING (*N*) is executed.

the block. Thus, the block will translate if the lightpen is moved radially with respect to the centroid and will rotate if the lightpen is moved tangentially with respect to the centroid (see **Fig. 1g**). A general trajectory of the lightpen will be decomposed into its radial and tangential components resulting in simultaneous translation and rotation. Any text related to the block will move with the block but will not rotate.

3. MOVE + *Me*—as long as the endpoint remains the intended second operand, the pin corresponding to the endpoint is removed and the endpoint follows the lightpen. All lines connected to the endpoint act as rubber bands (see **Fig. 1h**).
4. MOVE + *Mt*—as long as the text remains the intended second operand, it follows the lightpen. Hence, all hooks (rubber bands) connected to the text stretch.
5. MOVE + anything else—results in a DO-NOTHING (*N*).
6. Anything + MOVE—results in a DO-NOTHING (*N*).

The entire reference (modelling) space can be considered as a large virtual space with only a small portion of it displayed on the screen at any one time. Hence there must be some provisions for translating the virtual space thereby causing different portions of the space to be displayed on the screen. The translation is accomplished by selecting any item in the space to be translated (reference or modelling) as the first operand and a 'velocity box' as the second operand. The *velocity box* is a square containing a 16 × 16 grid. A velocity vector is defined within the square in the following way. The initial point of the vector is the centre of the square. The terminal point of the vector is defined by the position of the lightpen within the square. As long as the velocity box is the second intended operand (an item is an intended operand only while it is being pointed at by the lightpen), the entire reference (modelling) space moves with the indicated velocities.

If text is related to a block or line(s), the function called *lite* will cause the text to appear or disappear (see **Fig. 1i**). This enables the text to be 'turned-off' when the screen becomes too cluttered or 'turned-on' when it becomes necessary to examine or modify the text. The lite function is defined by selecting a light key labelled LITE as one of the operands. The other operand is the block or line(s) related to the text. If the text is currently turned on, this function will turn it off and vice versa. When related text is turned off, the hook that related the text is also turned off. (If one text is related to several blocks and/or lines, then this function merely turns on or off the relating hook but not the text itself unless all the other relating hooks for this text are currently turned off.) A hook relating a text and a block can also be turned on or off without turning on or off the text (see **Fig. 1j**). This is accomplished by selecting the light key labelled LITE as one operand and the text as the other operand. (If the text is related to several blocks, all relating hooks will be turned on or off by this function.)

The following operands will result in a DO-NOTHING (*N*) in all but a very limited number of cases. Thus, these operands have been omitted from the discussion thus far so as to simplify Tables 3 and 5. Two such operands (*Dv* and *Mh*) and the instances in which a function other than DO-NOTHING (*N*) is executed are listed here.

A. *Dv*—The dividing line between reference and modelling spaces

First operand =	MOVE
Second operand =	*Dv*
Function =	Move the dividing line thereby increasing or decreasing the room allowed for the reference or modelling space

B. *Mh*—Hook in modelling set

First operand =	*Mh*
Second operand =	*Rn*
Function =	Erase (*E*) a hook thereby undoing a relation that was created previously

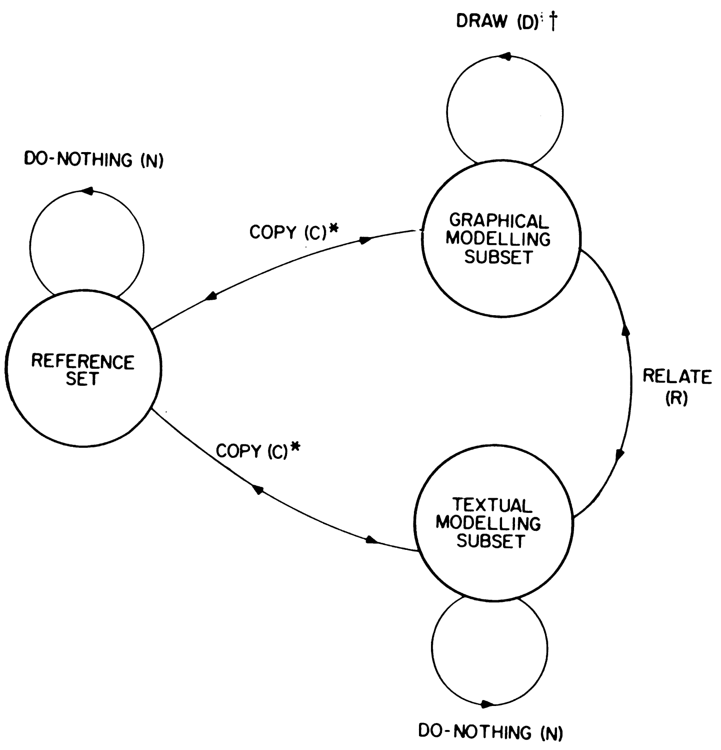

Fig. 3. Graph representation of relation between operands and functions with text considered

*If the operand in the reference set is a new point (*Rn*), an erase (*E*) is executed instead of a copy (*C*).
†If the two operands are the same, a do-nothing (*N*) is executed instead of a draw (*D*).

Filing and retrieving

The filing and retrieving scheme described here is unique in several ways. For one thing, it is the filing and retrieving mechanism itself that provides the ability for structuring models and for zooming in on fine details of the model. For another thing the filing and retrieving mechanism provides the ability to define new reference and modelling spaces thereby leading to a very simple bootstrapping technique for system generation. These features will become more apparent after the filing and retrieving mechanism is described.

Complicated models are often broken up into small parts in order to simplify the representation of the model. To provide this ability, some blocks may be defined as a set of interconnected blocks. Such blocks are called *complex blocks* and the interconnections of blocks that make up the complex blocks are called the *contents of the complex block*. Similarly the *contents of the reference or modelling space* is the interconnection of blocks currently being displayed in that space. If a block has no contents it is called a *simple block*. The *symbol of a block* is the pictorial representation of that block. The *name of a block*, on the other hand, is a verbal representation of the block. Note that the symbol of a complex block is usually not the interconnection of the symbols of the blocks making up its contents; rather it is a unique symbol pictorially representing the complex block.

Complex blocks are defined through the use of the *define* function. The define function is implied by selecting a light key operand labelled DEFINE. If the other operand is a block (*Mb* or *Rb*), the contents of the block are replaced by the contents of the modelling space. Complex text can similarly be defined. Any comments applying to complex blocks also apply to complex texts.

The contents of a complex block can be displayed in either the reference space or the modelling space. The corresponding functions that do this are the *display-reference* and the *display-model* functions. Note that the define and display functions are, in effect, performing filing and retrieving operations.

The display-reference or display-model functions are implied by selecting a light key operand labelled DISPLAY-REFERENCE or DISPLAY-MODEL respectively. If the other operand is a complex block (*Mb* or *Rb*), the contents of the reference or modelling space are replaced by the contents of the block. If the operand is a simple block, a blank reference or modelling space appears.

At all times there is displayed in a corner of the reference (modelling) space the symbol of the block whose contents are being displayed in the reference (modelling) space. This symbol is called the *current reference (modelling) space indicator.* Any succeeding changes made to the contents of the modelling space will automatically update the contents in the current modelling space indicator (note it is not possible to make changes to the contents of the reference space). Hence, an implicit filing is done whenever the contents of either of the spaces on the screen changes.

The symbol of the block, whose contents were displayed in the reference (modelling) space prior to the last execution of the display-reference (display-model) function, is displayed next to the current reference (modelling) space indicator. This symbol is called the *previous reference (modelling) space indicator.* The previous reference and modelling space indicators can be used to go up or down through the levels of a complicated structure as will be next demonstrated.

Each time a display-reference (display-model) function is executed, the symbol in the current reference (modelling) space indicator is put on a reference (modelling) push-down stack. However, if a display reference (display-model) function uses the symbol in the previous reference (modelling) space indicator as its other operand, the symbol in the current reference (modelling) space indicator is not put on the reference (modelling) stack and instead the last entry is removed from the stack and placed in the current reference (modelling) space indicator. The resulting last entry on the stack is put in the previous reference (modelling) space indicator and the contents of the new block in the current reference (modelling) space indicator are displayed in the reference (modelling) space. With the exception of the operation of the stack, the symbols in the current and previous space indicators behave as *Rb*'s if they are used as operands of any functions.

Thus, the display functions together with the previous space indicators provide the ability of going down through the levels of a complicated structure and returning up through the structure. In addition, attacher points on a copy of a complex block and text associated with the copy could be made to correspond to lines and text in the contents of the complex block. This feature is analogous to the transferring of arguments between a main program and its subroutines.

A complex block can be converted to a simple block by the use of the *remove-contents* function. This function is implied by selecting an operand labelled REMOVE-CONTENTS. If the other operand is a block (*Rb* or *Mb*), the contents of that block are destroyed. If the block is the current reference or modelling space indicator, a blank screen appears for the reference or modelling space.

A portion of the contents of a complex block can be added to the modelling space (i.e. added to the contents of the block whose symbol is in the current modelling space indicator) by use of the *add-contents* function. This is illustrated in **Fig. 1k.** In that example, the two triangles are the contents of the block represented by the circle. The add-contents function is implied by selecting a lightpen operand labelled ADD-CONTENTS. If the other operand is a block (*Rb* or *Mb*), the portion of the block's contents that appeared on the screen when the block was last displayed is added to the contents of the modelling space and is superimposed over the portion of the virtual modelling space that is currently displayed on the screen. This function gives the user the ability to copy the contents of a block in his model instead of merely copying a single block.

System organisation

As mentioned previously, the filing and retrieving mechanism allow for a simple bootstrapping technique for system generation. This will now be described.

All blocks that are ever used in either the reference or modelling space are contained in a library called the reference library. Hence, the reference library is the set of all blocks in the system. Through a separate phase, the user can enter symbols for new blocks into the reference library.

The system organisation described below provides a user with a bootstrapping facility for generating reference spaces; these spaces can then be used in future modelling. This bootstrapping facility means that the number of blocks provided by the system can be small and yet the system will be completely general.

At the time the system is created, the reference library contains only the following blocks:

1. MENU block—This is the block whose symbol appears in the current reference space indicator and whose contents appear in the reference space when the user signs on. (The symbol in the previous reference space indicator when the user signs on is the symbol of the block whose contents were displayed in the reference space when the user last signed off.)
2. WORK block—This is the block whose symbol appears in the current modelling space indicator and whose contents appear in the modelling space when the user signs on. (The symbol in the previous modelling space indicator when the user signs on is the symbol of the block whose contents were displayed in the modelling space when the user last signed off.)
3. LIB block—The contents of this block consist of all blocks, other than LIB itself, which are ever used in either the reference or modelling space. When the user adds symbols for new blocks to the reference library (through a separate phase), these symbols appear in the contents of LIB.

The contents of the above blocks at the time the system is created are shown in **Fig. 4**.

The effect of the define function is to associate the contents of the modelling space with a block in the reference library. The user can define any block in the reference library to be an interconnection of other blocks in the reference library with the following exceptions:

1. The user cannot define the block LIB.
2. The user cannot remove the appearance of the block LIB from the contents of the block MENU. This guarantees that the user will always be able to display the library in the reference space and hence get access to every block in the library.

New blocks are created (and modified) during a separate phase and placed in the reference library. Such new blocks can be used initially only by copying out of the reference library (LIB must be in the reference space). After the new block is copied into the modelling space, the contents of some other block can be replaced by the contents of the modelling space (via the define function) and the contents of that block can later be call-referenced thereby giving the user access to the new block again.

New texts are created and/or modified directly in the modelling space as an *Mt*. Text never gets entered into the reference library. After the text is created in the modelling space, the contents of some other block can be replaced by the contents of the modelling space (via the define function), and the contents of that block can later be call-referenced causing the text to appear in the reference space as an *Rt*. If that *Rt* is copied into the modelling space, it becomes an *Mt*. This *Mt* is independent of the *Rt* and can be modified without affecting the *Rt*. Also, if several copies of the same *Rt* are made, each copy can be modified separately.

Implementation

A subset of the modelling functions was implemented on an IBM 1130 with 8K words of 16 bits each. The 1130 uses a modified IBM 2250 Model III display console which provides the graphic output. The 2250 uses the 1130 main storage as a display buffer. Although the functions presented in this paper have been described in terms of light-pen input they are in fact independent of the actual input medium. In the present implementation, a Sylvania Tablet with electronic pen was used for the graphic input.

The functions implemented are COPY, DRAW, ERASE, DO-NOTHING, and SELECT. In addition, some primitive text functions were added so that parameters could be associated with blocks in the model. The modelling program can be connected to an application program that analyses the model created. One such application program included in the present implementation is the 'IBM 1130 Continuous Systems Modelling Program' which is capable of analysing an analog computer configuration and generating its response.

The users of the implemented system were able to properly execute the modelling functions after about 15 minutes of experimenting with the system and virtually no explanation of the system. Those users that were familiar with other drawing programs were indeed impressed with the ease of learning and ease of use of implicit functions in contrast to explicit functions.

Acknowledgements

The authors are grateful to W. A. Muenzner for his comments and ideas regarding text editing and for his Text Editing program (Muenzner, 1968) which was used to produce all working drafts and the final copy of this paper. The authors are also thankful to L. A. Belady for his encouragement and guidance throughout the research and his helpful suggestions and criticisms. The authors would like to thank H. B. Baskin for initially suggesting topological modelling as an area of research.

Appendix: Entering and modifying text

Text is entered as a string of characters from the alphanumeric keyboard. A terminal character in the string is used to delineate items of text. The initial position of an item of text is specified as one of the operands of the function that creates the text. Each item of text has a top, bottom, left, and right margin defined as follows:

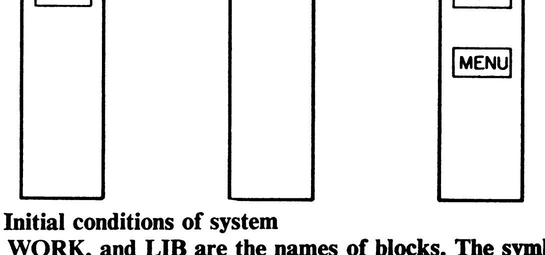

Fig. 4. Initial conditions of system
MENU, WORK, and LIB are the names of blocks. The symbols for the blocks are $\boxed{\text{MENU}}$, $\boxed{\text{WORK}}$, $\boxed{\text{LIB}}$ respectively. The contents of the three blocks are shown in the figure.

1. Left margin of text is the position of a vertical line passing through the initial position of the text.
2. Top of text is the position of a horizontal line passing through the initial position of the text.
3. Right margin and bottom of text are initially defined as right margin and bottom of virtual modelling space.

A cursor is used to indicate where in the modelling space the next typed character will appear. The cursor takes the form of an underscore in a character position unless the cursor precedes the left margin or follows the right margin of the text. In such cases, it takes the form of an arrow to the line of text. The cursor is advanced one character position each time a character is typed. In addition, the following four typewriter keys provide facilities for moving the cursor:

1. Spacebar—Move cursor right one character position but write nothing. If cursor was one character position beyond right margin, cursor goes to one character position beyond left margin on next line.
2. Backspace—Move cursor left one character position but write nothing. If cursor was one character position beyond left margin, cursor goes to one character position beyond right margin of preceding line.
3. Advance—Move cursor down one line and over to one character position beyond left margin. If cursor was at bottom of text, text will roll up one line.
4. Jump—Move cursor up one line and over to one character position beyond right margin. If cursor was at top of text, text will roll down one line.

Characters appear at the current location of the cursor and the cursor is advanced one character space each time a character is entered. If cursor precedes left margin, character appears at location of left margin (i.e. one character position to right of cursor) and cursor moves right two character positions. If cursor follows right margin and character (other than a space) is entered, all preceding characters up to last space are moved to left margin of next line and entered character is placed after these characters. Cursor is placed after entered character. All characters following the cursor are similarly moved to make space for these moved characters.

All text editing commands (with the exception of character replacement) are explicitly given by pressing an alternate-code key simultaneously with some other alphanumeric character key. For example, a delete command is given by typing *D* while depressing the alternate-code key. An alternate-code key is similar to an upper-case key in the sense that it gives special significance to any alphanumeric character. Unlike the modelling functions, it does not appear that the text editing functions can be implied by context but rather must be given as explicit commands. A list of a suitable set of text editing functions is the following:

1. Replacing characters—Overtyping any character has the effect of replacing the old character with the new character. A command exists for replacing a character with a blank (a space does not do this).
2. Delete character at cursor location—The delete command causes succeeding characters to be moved left one character position.
3. Insert characters preceding cursor location—The insert command causes future characters to be inserted preceding the current location of the cursor. Another command must be given to end the insert. A backspace typed while in this mode has the effect of undoing the last character inserted (i.e. deleting the character preceding the location of the cursor).
4. Delete line—This function is executed if delete command is given when cursor is located beyond left margin.
5. Insert line preceding current line—This function is executed if insert command is given when cursor is located beyond left margin.
6. Insert line following current line—This function is executed if insert command is given when cursor is located beyond right margin.
7. Define right margin—The define-right-margin command defines the vertical line passing through location of cursor as right margin for all future lines of text typed in. Present lines remain as they are. However, if text is turned off and then turned back on again, all lines of text will use the last defined right margin. Also any future copies of a text will use the last defined right margin.
8. Define bottom—This command defines the horizontal line passing through location of cursor as bottom of text. All lines beyond bottom of text are blanked. If advance key is pressed when cursor is at bottom line, text will 'roll up' one line.
9. Start new line—The new-line command creates a special symbol at the cursor location and all characters on that line that are to the right of this special symbol are moved to the beginning of the next line. The characters on this next line are moved to the right to make room for the inserted characters. Any words that in whole or part would be moved past right margin are instead moved to beginning of next line etc.
10. Removing right margin—This command defines the right margin to be the right side of virtual screen for all future lines of text typed in. Note that a new right margin can be defined even though a right margin has previously been defined. However, if the new right margin is to be to the right of existing right margin, it is impossible to position cursor at this location without first removing the previous right margin.
11. Removing bottom—This command defines the bottom of virtual screen as bottom of text. All lines beyond previous bottom line location and up until bottom of virtual screen are made visible. Note that a new bottom can be defined even though a bottom has previously been defined.

References

BASKIN, H. B., and MORSE, S. P. (1968). Multilevel Modelling in a Graphical Design Facility, *IBM Systems Journal*, Vol. 7, Nos. 3 and 4.

ELLIS, T. O., and SIBLEY, W. L. (1967). On the Development of Equitable Graphic I/O, *IEEE Trans. on Human Factors in Electronics*, Vol. HFE-8, No. 1, pp. 15-17.

HORNBUCKLE, G. D. (1967). The Computer Graphics User/Machine Interface, *IEEE Trans. on Human Factors in Electronics*, Vol. HFE-8, No. 1, pp. 17-20.

MUENZNER, W. A. (1968). An Experimental Text Editor for the IBM 1130/2250, *IBM Research Report RC*-2063.

SUTHERLAND, I. E. (1963). Sketchpad—A Man Machine Graphical Communication System, *AFIPS Conference Proceedings*, SJCC 23, pp. 329-346.

SUTHERLAND, W. R. (1966). The On-Line Graphical Specification of Computer Procedures, Ph.D. Dissertation, MIT, Cambridge, Mass.