DESIGNING A LANGUAGE FOR INTERACTIVE CONTROL PROGRAMS

M.A. Maclean

Communications Research Centre
Department of Communications
Ottawa, Ontario

Introduction

Man-machine interaction through the medium of a graphical display is becoming more widespread as the cost of hardware drops, and it is clear that many possibilities exist for fruitful applications in a variety of fields. This two-way communication process can involve messages in any form but it is commonly done by having the computer write phrases on the display and allowing the human operator to indicate his choice of one of these by pointing to it with a light-pen or moving a cursor to lie close to it. When he does this, the machine is programmed to take some appropriate action and the overall result of this ongoing dialogue is to guide a complex process through a sequence of decision points that require human judgment to resolve.

This style of interaction has spawned its own vocabulary. For example, the messages displayed by the computer are referred to as 'light-buttons' and collectively as a 'menu'.

The computer may display other kinds of information as well. Very often the purpose of the interaction is to build or modify a stored model of some situation in the real world; e.g., an electrical schematic, an engineering drawing or an architectural plan. Or the output of a computation might be displayed in the form of a family of curves for different values of a stepped parameter. In either case we can imagine an interactive process in which the operator wishes to draw attention to certain displayed objects and to have specific actions performed as a result. A scheme of classification is necessary here, since the precise content of the displayed picture will usually not be known when the program is written. The generic types of entity displayed in a picture will be referred to as 'picture parts' and the individuals as 'instances'. Thus, in an electrical schematic, a resistor called R1 would be an instance of the generic type 'resistors'.

Of course there is no intrinsic difficulty in programming a computer to display a series of messages and pictures or in branching execution to specific routines when light-pen strikes are registered. However, the programs can be long and tedious to write in conventional computer languages and difficult to modify quickly when new features are wanted in an existing interactive system. Moreover, many display systems (e.g., storage-tube displays) require the entire display to be drawn before searching can take place for an operator intervention. This adds complexity to the program by forcing a separation between the display and search operations. Other hardware-dependent factors impose their own logic on the program with the result that interactive systems are difficult to move to a new machine.

One purpose of this paper is to show how a standard program model, with its logical structure embodied in a special-purpose language, can bring some relief in this area. A second is to show an example of a technique for implementing such a language which is fast, flexible, and easy to use on a wide variety of computers.

Development of the Approach

The initial impetus for the development of the language was the desire to simplify the building of interactive programs by making use of man-machine interaction in the design process itself. It was intended that light-button menus should be created 'on-line' (see, for example, reference 1) to avoid the tiresome chore of specifying screen layout as a series of calls to standard graphical subroutines. Furthermore, it was hoped that the other component of an interactive system could be specified in the same way, that is to say, the actions required of the system in response to operator intervention. The question then arose--"What would be the output of this design process?" A machine-language program or a set of tables have the disadvantages of being machine-dependent, unreadable, and difficult to modify. A much better method would be to punch out a description of the result of the design process in an intermediate language. A compiler would then translate this to an executable program. This technique has a number of advantages:

- the result of the design process is recorded in palatable form,
- the program at this stage can be modified easily,
- the intermediate-language program can be translated for any suitable interactive display system. Thus the design can be carried out on one system for use with another.

Once this was realized, it became obvious that the first step should be the implementation of the language and its translator so that the language itself could be used as a tool in the building of the design system. Indeed, the language is proving very useful in its own right and greatly reduces the labour of writing interactive programs.

Interactive Control Programs

Consider the structure of an interactive system involving a CRT display as the man-machine interface (Figure 1). In this example the purpose of the system is to build and edit a data-base and to carry out computations and I/O operations under the direction of a human operator.

The system is shown as a collection of specialized functions, each being realized by one or more routines having jobs such as data-base editing, data extraction, numerical computation, data display, and so on. It is likely that many of these routines will be standardized from one application to another and can be drawn from a library that grows as new specialized functions are implemented.

The routines that generate the light-button menus and those that interpret the light-pen strikes and control the execution of the specialized

functions, however, are peculiar to each application and contain code that is highly dependent on the structure of the display and light-pen system. It is this latter class of routines which will be referred to collectively as the 'control program' for an application. It is convenient to include with this any initialization which has to be done when the system is started up, as well as the declaration of global variables, communication with the monitor, and so on. In other words, the control program is host for the entire system.

Of course the display is not the only possible output channel for messages, neither is the light-pen the only possible input. Other input devices such as push-buttons are appropriate but the language has not yet been extended to include them.

The Control Program Language

The overall aim is to present the designer of an interactive system with a packaged 'construction kit' for control programs that has as much flexibility as he can usefully employ and yet which removes as much burdensome detail as possible from his work.

The logical structure of the control programs is shown diagrammatically in Figure 2, and an example of a simple program is given in Figure 3. The contents of the display at any particular moment constitute a 'page' which may include 'light-buttons' which have been generated by the control program and 'pictures' displayed by external routines. There can be any number of pages. When a particular page is displayed, the display routines are executed first and then the computer loops through a routine which looks for a light-pen strike. If a strike is detected, another routine tries to identify the object pointed to by the operator. If it is successful, control is passed to an 'action routine' which ends either with another search for a light-pen strike or in the displaying of the same or another page. If the strike cannot be identified, the system waits for another strike.

The light-buttons are displayed by the control program itself but the display of pictures and the performance of complicated actions following a light-pen strike are carried out by external routines. Thus the language has a statement type whose purpose is to call an external routine with any number of parameters.

The general form of a control program consists of the following: (see Figure 3)

a) a declaration of the name of the program;

b) an entry sequence which must at least specify which page is to be displayed first, but which generally initializes variables as well;

c) one or more page descriptions consisting of a page number declaration, descriptions of light-buttons and pictures to be displayed, each followed by any actions that must ensue when a light-pen strike is registered, and finally an END OF PAGE statement;

d) an END OF PROGRAM statement.

The description of a light-button is started with a LIGHT BUTTON statement and consists of the statements necessary to display the button followed by the action routines to be invoked by that button. A light-button may consist of a number of separate objects (e.g., an alphabetic keyboard) which are prefaced by statements ITEM 1, ITEM 2, and so on. When the light-pen sees a particular one of these, the item number is stored in a variable called ITEM which may be referenced for branching purposes or passed to an external routine.

The branching to the various action routines is controlled by integer variables, here V1 and V2, which may be created as required and given arbitrary names. This form of branching allows the user to represent the logic of his program as a matrix where the rows are the light-button or picture-part numbers and the columns are values of a variable. It can also be looked upon as an automaton having a 'state diagram'[2] whose changes of state are triggered by light-pen strikes (Figure 4). Multiple variables are allowed since interactive systems are often most easily represented by two or more processes occurring at the same time. The form of the ACTION IF . . statement makes it easy to specify processes where a particular light-button may initiate transitions from more than one state. Any number of ACTION IF . . statements may be used with a light-button. There is also an unconditional ACTION statement.

A picture is displayed by calling an external routine with a PICTURE . . statement. An external routine is also responsible for identification of the entity in the picture seen by the light-pen since, of course, the detailed structure of the picture is not known to the control program. The identification is specified by a statement of the form SEARCH PICTURE WITH . . . Certain conventions must be observed in constructing the search routine. It has three parameters which are, respectively,

the input to the search

- the light-pen strike number, counting from the start of the picture

and the output

- the picture-part number, according to some scheme established by the user

- the item number, which is a sub-classification similar to that used for light-buttons and corresponds to the particular instance pointed to.

If the search routine cannot identify the strike as belonging to the picture; i.e., if there are not enough identifiable entities in the picture, it must return the negative of the number of entities it actually contains.

Following the search statement is the specification of the desired actions in the form

```
PICTURE PART 1
ACTION IF V1 = 1,3,X,4
. . .
. . .
ACTION IF V1 = 5
. . .
PICTURE PART 3
ACTION IF V2 = 1
  etc.
```

As with the light-buttons, the variable ITEM contains the item number of the strike if it is needed.

The action routine specified in the control program may include elementary assignment operations upon the integer variables and calls to external routines which may use any of the variables or the special variable ITEM as parameters Each action routine must end with a SEEK statement, which causes the system to look for another light-pen strike in the current page, or a DISPLAY . . statement which triggers the display of one of the pages specified in the program.

One may wish to have the display of a particular light-button or picture conditional on the value of a variable. This is expressed by statements such as

```
LIGHT BUTTON IF X = 2

or PICTURE R17 IF V1 = V2
```

Another type of conditional statement is probably desirable for testing the value of a variable returned by an external routine. However, this latter feature has not yet been implemented.

Program Structure

The structure of the program in memory is a combination of straight-line code, tables and list-structures. Certain functions, such as light-button identification and the decision procedures involved in choosing a course of action after a light-pen strike, are best handled by fixed routines which use tables or parameters produced by the translator. Because the ordering of statements in a control program is one designed for clarity and ease of use rather than for simple sequential translation, and partly because of the inherent complexity of the program, the translated program takes the form of a number of segments connected by pointers.

As the contents of each page are described in the source language, two intertwined chains of code are generated (Figure 5). One consists of the code to display the light-buttons and pictures, and the other contains the calls to the identification routines and the tables which describe the logical structure of the page (i.e., the number and ordering of the light-buttons and picture-parts and pointers to the action routines). The action routines are chained by pointers for each light-button or picture-part and each contains a decision table to determine whether or not it should be executed. The physical location of the action routines is at the end of the translated program.

Translation

This is a two-step process. The first uses a general purpose

macro-processor called STAGE2 which translates the control program language into assembly code for the target machine (in this case a DEC PDP-9) and establishes the program structure of Figure 5. The assembler is used for the second step.

STAGE2 is a macro-processor with many powerful features that has been designed as part of a programming system to be easily implemented on almost any computer(3)(4). Apart from a small host program and a simple I/O interface, it is written in the language of a pseudo-machine which can be translated with a rudimentary macro-processor that is coded by hand or in FORTRAN. This bootstrapping process makes it possible to place STAGE2 in operation without access to a working version on another machine.

The task of writing the translator consists of building a set of macro definitions which are read by STAGE2 before the input of the source language program. All the processing done by STAGE2 is character-oriented and results in the output of a character string; i.e., an assembly-language program for the target machine. This step could be carried out on any machine having the same set of character codes.

Each statement in the control program language is a macro call and one of the attractive features of STAGE2 is the flexibility of format which is allowed. Parameters can be of any length and can be arranged in any way within a 'template' of fixed characters. By making it possible to interpret strings such as 1,4,15,V1,3... as a single parameter, and providing mechanisms for scanning such strings and picking out the components, it is very easy to translate the ACTION IF V3 = 2,3,X,5 type of statement.

Some of the translation is very straightforward, for example the simple assignment statement V1 = 2, while other statements invoke complex mechanisms which are responsible for setting up the structure of Figure 5. A certain amount of this work is passed on to the assembler. For example, the pointer chains in Figure 5 are created by using symbolic references to labelled locations. To a considerable degree these mechanisms are machine-independent and so this part of the translater would not have to be rewritten to move the whole system to another machine. The part which actually generates code for the target machine's assembler would, of course, have to be changed.

At the time of writing, no diagnostic features have been provided in the translator. This is partly because of the experimental nature of the language but it is expected that memory limitations will make it difficult to do the necessary checks at the time of translation. A solution to this problem is to submit the source program to a diagnostic processor prior to translation. This would also use STAGE2 but with a different set of macro definitions. Another desirable improvement would be to free the input format from the rigid indenting conventions and permit the user to use spaces freely.

The power and flexibility of the STAGE2 macro-processor greatly speed the writing of a translator of this sort and encourage the development of special-purpose languages. The first version of the control program language, simpler than the present one, was written in four days and the features described here were developed over a further period of about three weeks. New features can usually be added rather quickly unless they involve radical changes in program structure.

Conclusion

We can summarize as follows: a well defined area has been outlined (control programs for interactive graphical display systems) where a standard program model is deemed to be able to satisfy a broad class of applications. Then a language has been created which exemplifies the logical structure of the program model, as seen by the user, and which he can use to express his particular requirements without the burden of unnecessary detail and, most important, without the danger of making trivial errors which can cause obscure side effects. The translator now produces an executable program according to the general form, but with detailed variations of structure to suit the particular case.

A common objection to special-purpose languages is that translators require a lot of work to write. This need not be so if the proper tools are available. If, in addition, the tools are constructed in such a way that they can be moved readily from one machine to another, the objection has very little weight.

Acknowledgement

The writer is indebted to H.G. Bown and C.D. Shepard for helpful dis-discussions during the project and the writing of this paper.

References

1. B.W. Boehm, V.R. Lamb, R.L. Mobley, and J.E. Rieber, 'POGO: Programmer-Oriented Graphics Operation' Proc. SJCC 1969, pp. 321-330.

2. W.M. Newman, 'A System for Interactive Graphical Programming' Proc. SJCC 1968, pp. 47-54.

3. W.M. Waite, 'Building a Mobile Programming System', Computer Journal 13 no. 1, February 1970, pp. 28-31.

4. W.M. Waite, 'The Mobile Programming System: STAGE2', Comm. ACM 13 no. 7, July 1970, pp. 415-421.

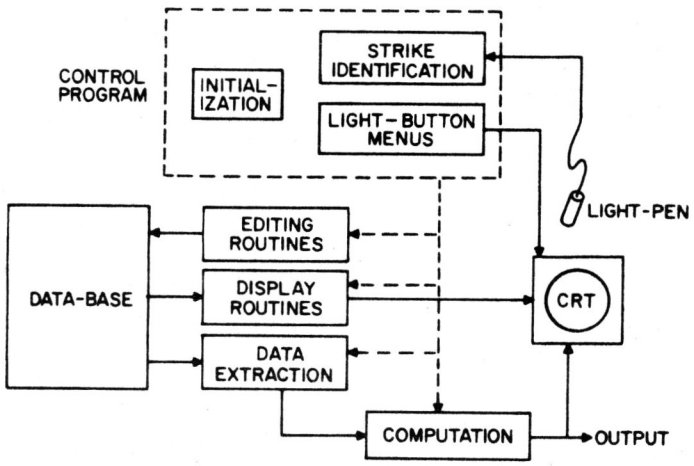

Figure 1 Structure of an Interactive System

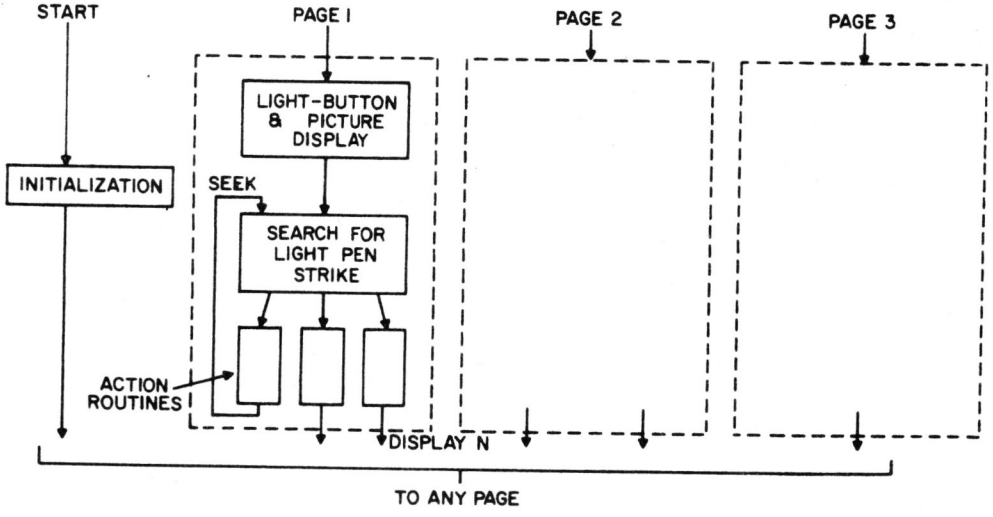

Figure 2 Control Program Logical Structure

```
CONTROL PROGRAM WXYZ
   V1 = 1                          } initialization
   DISPLAY 1                       }
PAGE 1
 LIGHT BUTTON                      }
  ITEM 1                           }
   BEAM 1500,1500                  }
   WRITE UP                        } display generation and
  ITEM 2                           } action specification
   BEAM 1500,1400                  } for one light-button
   WRITE DOWN                      }
  ACTION IF V1 = 2                 }
   V1 = 1                          }
   DO R1(ITEM)                     }
   DISPLAY 1                       }
 LIGHT BUTTON
   BEAM 1500,1600
   WRITE MOVE
  ACTION IF V1 = 1,3
   V1 = 2
   SEEK
 PICTURE R2
 SEARCH PICTURE WITH R3
  PICTURE PART 1                   } action for one category
  ACTION IF V1 = 1,2               } of object in picture R2
   V1 = 3                          }
   DO R4(V2)                       }
   DISPLAY V2                      }
END OF PAGE
PAGE 2
 . . .
 . . .
END OF PAGE
END OF PROGRAM
```

Figure 3 Typical Control Program

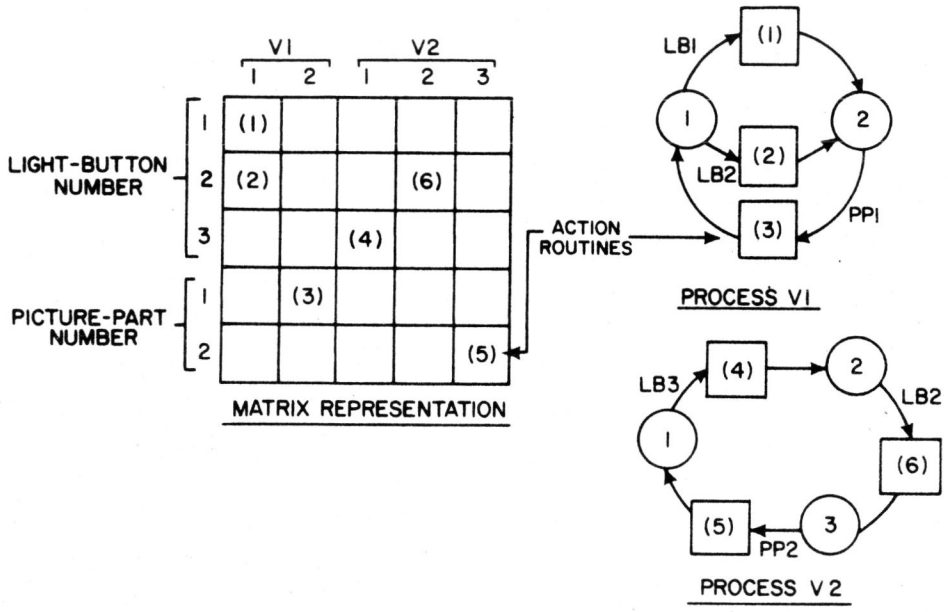

Figure 4 Alternate Representations of Control Processes

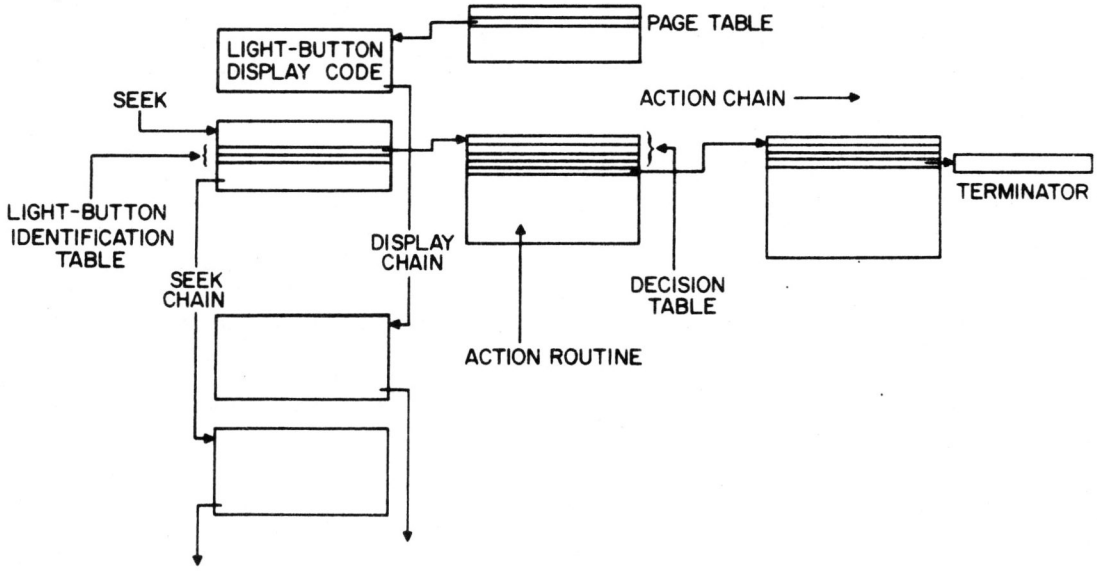

Figure 5 Control Program Structure