A Graphically Interacting Program Monitor

B. E. J. Clark
Gould SEL Computer Systems Ltd., Raffety House, 2–4 Sutton Court Road, Sutton, Surrey, UK
S. K. Robinson
Dept of Computer Science, Brunel University, Uxbridge, Middlesex, UK

This paper describes the implementation of a graphically interactive program monitor. It discusses the concept of control flow monitoring, and shows how this may be enhanced by the use of a graphic interface. It specifies the form of the graphic representation, and describes how the graphic display is derived from the program being monitored. It shows how the level of detail displayed may be controlled by the user to produce an optimum display.

1. INTRODUCTION

Sutherland[1] first proposed the use of graphics to display to a programmer the mechanisms of his program as it ran. Flowcharts have, for a considerable time, provided a static means for showing the flow of control through a program. Dijkstra[2] and a succession of structured methodologists have formalized the concept of program control mechanisms and described numerous chart representations of control flow mechanisms. Until recently, however, the high cost of both the graphic terminal and the large data storage needed to hold a graphic representation of a program have combined to prevent the realization of Sutherland's suggestion.

The sections that follow describe an attempt to exploit the rapidly falling cost of hardware in the production of a monitor with a graphic interface. Section 2 shows how the flow of control through a program is represented, and how a user is able to control the amount of detail he sees. Section 3 gives a brief description of the first implementation of these ideas using PASCAL as the host language, whereas Section 4 assesses its effectiveness by showing and discussing photographs of the output of the system. Section 5 concludes the paper by reviewing the initial system, and makes suggestions for future work in this field.

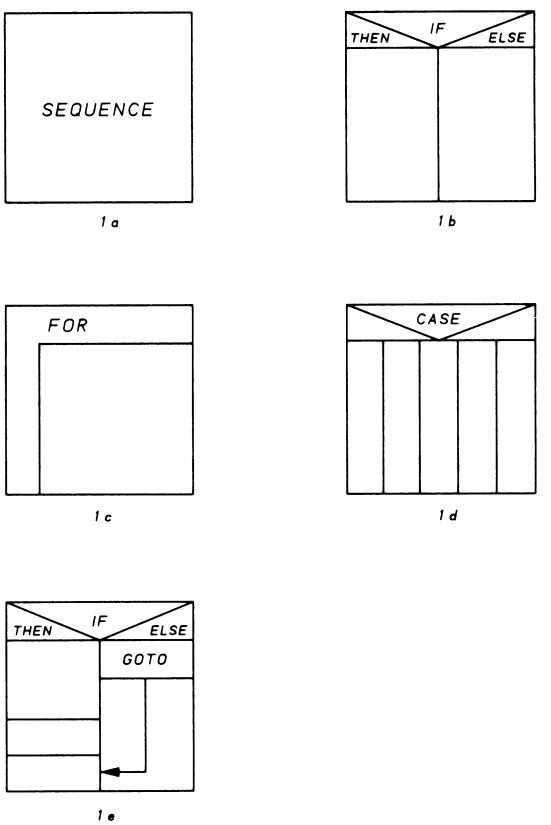


Figure 1. Flowchart symbols.

2. CONCEPTS

Plattner and Nievergelt,[3] in their review of execution monitoring systems, showed that the basic mechanisms have remained unchanged for over 20 years. The techniques, including breakpointing and trace statements, have been used to determine the flow of control through a program by implication rather than by direct observation. The object of the system described in this paper is to provide a user with a dynamic display of the flow of control through his program, rather than a series of monitor output statements. To achieve this aim the monitor process derives a pictorial representation of the program from its control structure. This is drawn at run-time, and the current locus of control is marked to indicate the progress of control through the program.

The diagrams used to represent the control structure are based on those defined by Nassi and Schniederman,[4] and are shown in Figs 1(a)–1(e). These symbols are combined to form an overall chart for a program such as is shown in Fig. 4. The rectangular shape of the overall chart is particularly suited for display on a graphic monitor. The method of constructing an overall chart by adding symbols within already existing symbols allows the monitor to define its maximum dimensions at the outset of processing, and to then fit the internal detail within known limits.

Control over the amount of detail shown on the screen is exercised by considering each statement as existing within a context. This context is determined by the surrounding statements; for example an assignment statement might exist in the simple context of a program or within the complex context of a conditional statement. At run-time the statement context is compared with the

CCC–0010–4620/83/0026–0235$02.00

user selected context to determine if the statement detail should be added to the diagram being shown. Four basic user levels of context are defined, namely:

0. No internal detail of a block
1. Structure only
2. Structure plus control statements (e.g. interative control)
3. All statements.

A call statement at level 3 will produce a level 0 display of the called procedure. Further increase in context level will result in the internal detail of the called procedure being displayed as a fresh display on the monitor screen.

3. MONITOR SYSTEM DESIGN

The system comprises a preprocessing analysis stage, followed by a compilation, after which the revised object program is able to call the monitor process. Communication between the analysis and monitor processes is via the data sets described in Section 3.1 below. In describing the monitor system design it is appropriate to first describe the nature of the data passed between the two processes, before discussing how it is generated and used.

3.1. Data design

Three sets of data are set up by the analysis process. These are the statement data set, the graphic data set and the global reference data set. Taking these in order, the statement data set is a compressed representation of the source statements used to create the other two data sets. It is used by the monitor process to add detail to the graphic representation in response to a user demand for increased detail.

LABEL	STATEMENT NUMBER	STATEMENT TYPE	STATEMENT CONTEXT	CALL NAME	STATEMENT START	STATEMENT LENGTH

Figure 2(a). Graphic data record.

Figure 2(a) shows the layout of a record in the graphic data set. Each of these records represents a single statement in the source program, and supplies the information necessary to draw that statement as a graphic symbol. Within each record the individual fields have the following significance:

The 'LABEL' field states whether this statement was labelled either as the target of a jump or as a component of a switch construct. In both cases this knowledge allows the monitor process to assess the scale of a control construct.

The 'STATEMENT NUMBER' field provides a unique key to the record, by referring it to the relative position of the source statement in the source program.

The 'STATEMENT TYPE' field provides specific information to the monitor about the form of the graphic symbol to be drawn for this statement.

The 'STATEMENT CONTEXT' field provides control information by indicating the level of context encompassing the specific statement.

The 'CALL NAME' field is used when control is passed to a procedure. It identifies the target of the transfer, allowing the monitor to search for the target block by using the global reference data set described below.

The 'STATEMENT START' and 'STATEMENT LENGTH' fields are used by the monitor to locate the character string representing the statement in the statement data set described above.

The third data set maintained by the analyser is a fixed length table of global reference records. The format of these records is shown in Fig. 2(b). The records are used to associate a 'NAME' with a particular 'ENTRY POINT' in a specific 'PROGRAM NAME'. Each entry in this table represents the target statement of a call within the program. The reference data from all programs analysed is set into this single reference table. This enables the monitor not only to trace control flow movement within a single program, but also to trace flow through independently analysed and compiled modules.

NAME	PROGRAM NAME	ENTRY POINT	ENTRY TYPE

Figure 2(b). Global reference data record.

3.2. Analysis process

The analysis process takes the input source file and parses it to derive the data necessary to generate the data sets outlined above. The action of the analyser is similar to that of a compiler in that a search is made for specific syntactic tokens that delimit the structured constructs illustrated in Figs 1(a)–1(e).

In the system implemented, Pascal was chosen as the source language, firstly for its well defined structured constructs and secondly for the ease with which the structural significance of a statement may be established. To further simplify the design it was decided to assume that the input source programs were free from syntax errors. Under these conditions it was only necessary to consider the 'PROGRAM', 'BLOCK' and 'STATEMENT' definitions of the language[5] to ascertain the necessary information to create the graphic and global reference data sets. Analysis is completed in a single pass over the source program, with all data generated on a statement by statement basis.

As well as deriving data from the source program, the analysis process also inserts into the source program calls to the run-time monitor process. A call is inserted following each executable statement in the source, referencing the statement number of the statement just parsed. Care is taken to ensure that the syntax of the inserted calls is correct. The revised program source is then submitted to the system Pascal compiler.

3.3. Monitor process

The monitor process is activated by a call inserted at the beginning of the revised source program. This initiates a

two stage analysis of the graphic record set. The first pass estimates the scale of a program block when displayed on a monitor screen. The second uses this scaling data to create a structure of linked circular lists, each entry being graphic data representing a program block in which the positions of each symbol on the screen are absolutely defined. Each element in an entry describes the attributes of a symbol on the screen. The lists themselves are linked together to allow the monitor to follow the transfer of control between blocks.

Once this structure has been established, control reverts to the program being monitored. At each statement a call is made to the monitor. This identifies the current statement number. The monitor checks the user keyboard to ascertain whether any changes to the display are required. If there are, these are actioned, and then a marker is placed over the symbol representing the last statement to show the passage of control through it.

Despite its experimental nature the design of the monitor system has proved very flexible. The use of data sets for communication has removed any knowledge of the source language from the monitor process. Given a change in the analysis process one could monitor any high level language in which the control structures were identifiable.

4. ASSESSMENT

There are two aspects to be considered when assessing the performance of the monitor system. The first is its accuracy in converting a source program into the specified graphical representation. The second is a more subjective examination of its effectiveness as a programming tool.

The accuracy of the monitor system has been established by comparing the output produced, in response to a given input source program, to a manually derived diagram of the same program. These checks highlighted the human ability to perceive the whole context of a diagram, and to make adjustments to produce a more balanced representation than can be produced by a purely mechanical process.

As has been stated, the assessment of the monitor as a software tool is far more subjective. Some idea of the effectiveness of this technique may be gained by the examples of monitor output taken at a graphic terminal. These examples are given as Figs 3(a)–3(c).† The following notes explain the display at each point in terms of the user requests to the monitor process.

Figure 3(a)

The context level has been set to 1 and the skeleton of the control chart is shown. The user can observe control passing from statement to statement.

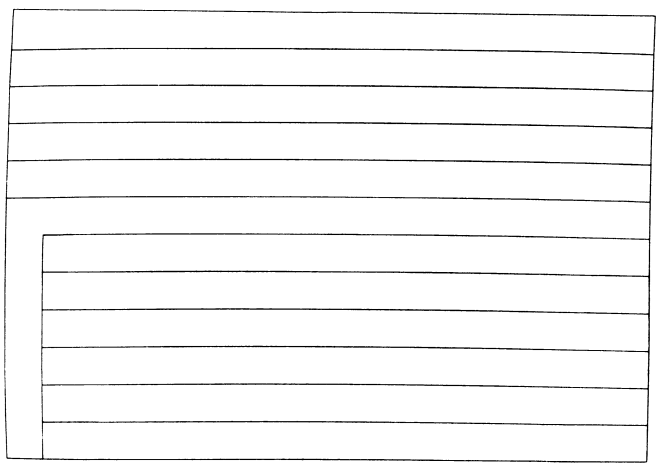
Figure 3(b)

Context level 3 has been selected and shows the whole of the program in terms of its statements. The flow of control through the program is marked at each statement encountered.

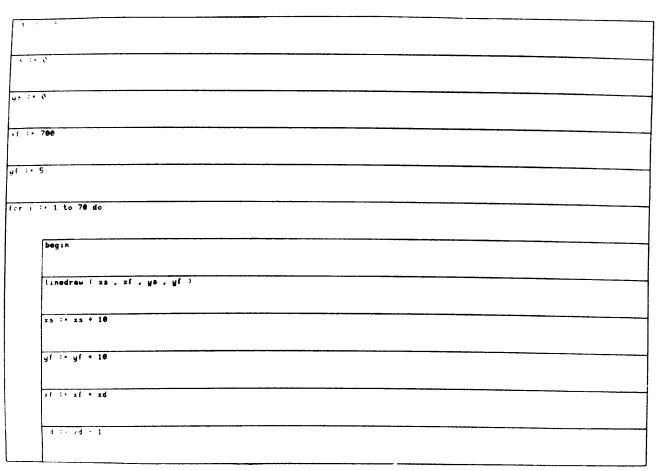
The control flow now encounters a call to a procedure ('linedraw' in the example). The result is to draw the procedure at a context level of 0.

Figure 3(c)

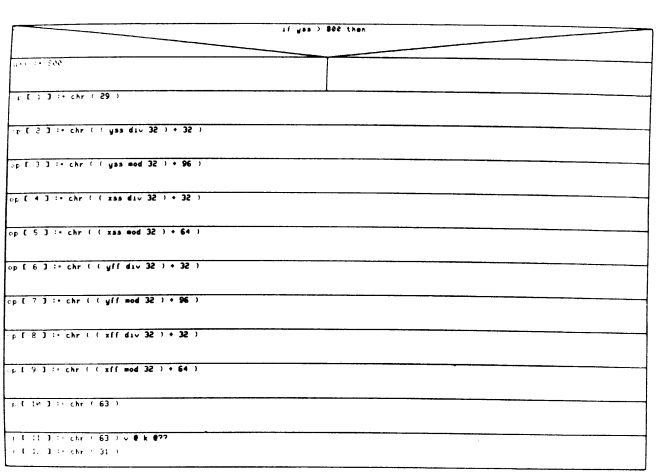
Further increases in the context level result in increasing detail of the internal structure of the procedure being



(a)



(b)



(c)

Figure 3

† Please note that owing to printing difficulties Figs 3(a)–(c) and Fig. 4 have been reversed out. The monitor background is in fact black and shows white lettering.

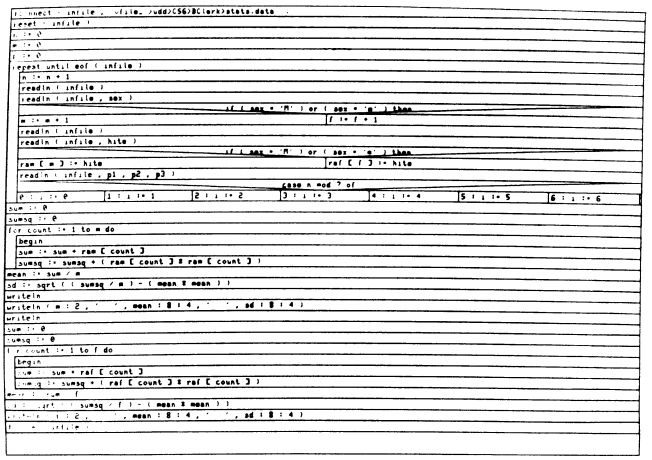


Figure 4

displayed. When control reverts to the calling process, the diagram of that process is redrawn at the context level current at the time of the procedure call.

These diagrams give a flavour of the control one can exercise over the monitor system. The more complex display, shown in Figure 4, illustrates the ability of the monitor system to show the structure of a more realistic program, in this case an undergraduate programming exercise. It should be stressed that, at this time, the system described is viewed as an addition to, rather than a replacement for, existing execution monitors.

5. CONCLUSIONS

The successful implementation of a graphic interface to a running program has added a new dimension to program monitoring techniques. The initial system, implemented at Brunel University, has been demonstrated to a number of potential users, from differing computer backgrounds. They have all found the diagrams easy to comprehend, and have been quick to relate the use of the system to their own environment.

Experience gained during the development of the system leads us to conclude that the structural analysis of the source program should be carried out in the compiler. The information required for monitoring is all extracted during compilation, albeit for a different purpose. By incorporating this analysis with already existing techniques for the extraction of monitoring data one can produce the data required in a single process. Also, the incorporation of analysis at the compiler stage would allow a finer level of monitor tracing to be established, by allowing analysis to take place at a level below that of a statement.

The effectiveness of the monitor process can be judged by the output shown in Figs 3 and 4. The Nassi–Schniederman symbols provide an ideal mechanism for using the graphic representation. Some minor adjustments could be made, but overall the chosen technique proved successful. Future development of the monitor process is likely to concentrate on a refinement of the display in terms of context and the application of the technique as a whole to other languages such as Ada.

REFERENCES

1. I. Sutherland, Computer graphics—ten unsolved problems. *Datamation* **12**, 22–27 (1966).
2. E. W. Dijkstra, Notes on structured programming. *EW249*, Technical University Eindhoven (1969).
3. B. Plattner and J. Nievergelt, Monitoring program execution—a survey. *IEEE Computer* **14**, 76–93 (1981).
4. I. Nassi and B. Schniederman, Flowchart techniques for structured programs. *SIGPLAN Notices* **8**, 12–26 (1973).
5. K. Jensen and N. Wirth, *Pascal. User Manual and Report*, Second Edition, Springer-Verlag, 116–118 (1978).

Received October 1982