# GPSS10: INTERACTIVE GPSS FOR THE DECSYSTEM-10

M. David Martin, Department of Computer Science
The University of Western Ontario, London
Ontario , Canada

## INTRODUCTION

In 1972 the University of Western Ontario replaced its IBM 7040 computer with a machine for which no GPSS software was then available. Since this language is used in teaching simulation techniques and in research studies at Western, the fact that GPSS was no longer readily available caused some inconvenience. The implementation of a GPSS processor for the University's existing DECsystem-10 timesharing computer was undertaken jointly by Mr. D. Link and the author in order to fill this gap in our software portfolio.

## THE IMPLEMENTATION

Our initial goal was to implement a subset of GPSS/360 which would be sufficient for students using GPSS in an "Introduction to Simulation" course. We wished to allow for expansion to the full GPSS language and for addition of interactive features, but the watchword of the project at this time was "Keep it simple". For this, and other reasons, we divided the GPSS system into three segments and, although details have changed, this overall design has been retained to date. The first segment is a fairly conventional symbolic assembler, producing a binary image of the source GPSS code with block opcodes represented as offsets in an opcode dispatch table and Standard Numerical Attributes represented as offsets in an SNA dispatch table. This binary image is passed in a disk file to the second segment which is the heart of any GPSS system -- the GPSS interpreter, front-ended by a simple loader. When the interpreter produces a report, it simply does a selective "dump" of internal tables to a disk file. This is later read by the third segment of the system and a standard GPSS report is produced from it.

## THE INTERPRETER

Before describing some features of the current implementation, a comment on the original version of the interpreter may be of interest. In that system a fixed quantity of each entity was provided, the number determined when the system was built. In view of the subsequent interest shown in the project by several sources, it was fortunate that the implementation language used (1) provided a way of making structuring of data independent of algorithms using that data. In this way, such unfortunate choices of data structures could be improved without the extensive and painful re-writing often necessitated by such changes. This is a technique which may be used, even with a good macro assembler (2).

## STORAGE LAYOUT

Storage layout for entity information is probably the most crucial data structuring area in a GPSS interpreter. These structures must facilitate rapid, random access, minimize wasting of space and yet allow run-time reallocation of numbers of entities and dynamic creation of entities such as facilities and queues. In GPSS10, entities are divided into two classes with regard to space requirements. Savevalues and logicswitches require half, one or two words each; transactions, storages and other entities require significantly larger amounts of memory. GPSS10 allocates space for the smaller entities as vectors of contiguous cells of the required size. Larger entities are allocated a one-word header cell each. Header cells for each type of entity are in a vector. When an entity is defined or referenced, a block of sufficient size is allocated to hold its information and the address of this block is placed in the corresponding header cell. This scheme permits convenient and random access to entity information and avoids reserving large amounts of storage for entities which might never be referenced in a particular model, since a zero value in a header cell indicates that the entity has never been referenced and has no other storage assigned to it.

In order to implement the REALLOCATE feature, it was necessary to allocate the header vector (or information vector, for the smaller entities) at run-time and to moor each one to a fixed two-word foundation cell for that entity type. One of the front-end loader's first tasks is to set up the entity vectors using either default or reallocated numbers of entities. It should be pointed out that while on some computer systems the saving of space resulting from this kind of vector scheme might not be a real one, current DECsystem-10 operating systems permit running programs to request more storage in increments of 1K words. Thus, extra space need not be reserved in a COMMON pool to allow for the case when say, all 500 facilities might actually be used by a model. Rather, a storage management algorithm (3) is used which takes advantage of this operating system feature to use only the amount of core storage needed by the model.

## FASTER EXECUTION

Several factors contribute to faster execution times of models run by GPSS10. Events chain pointers, for instance, are kept as memory addresses rather than as transaction numbers. Delay chain pointers use the DECsystem-10's capability to access bytes of arbitrary size by indicating the scan status bit of the delayed transaction. This allows delay chains to be cleared very rapidly. The block opcodes are divided numerically into two groups. Those blocks that can never refuse entry to a transaction are in one group, while TEST, GATE, SEIZE, ENTER and PREEMPT are in the other group. This reduces the overhead of deciding if a transaction can move from one block to another -- a very good spot to make even a small saving.

## INTERACTIVE FEATURES

The overall design philosophy of the interactive section of GPSS10 is one directed toward validating and debugging of models in a well-controlled and convenient way. This is in contrast to the direction taken by implementors of other interactive GPSS systems in which a model may be monitored and summary information is output, often using a graphics terminal to display histograms and the like. In GPSS10, the amount of I/O activity to the user's timesharing terminal is kept low. The simulation report may be directed to a device more suited to handling a larger quantity of information.

The prime requirement of an interactive simulation system is that it permit the execution of the model to be interrupted and later continued. A common method is to allow the user to interrupt the simulation by pressing a special key on his terminal. This is useful in only a limited way, however, due to the fact that the user has no real control over the state of the model when the interrupt occurs. Another method is to implement a special block which causes the simulation to be interrupted whenever a transaction enters it. In GPSS10 this is the PAUSE block. It is a great improvement over the less precise keyboard interrupt method, but the user must remember to place a PAUSE at each point where he wishes a pause to occur. If, during execution of a model, he wants a pause to occur in a place where there is no PAUSE block, he would normally have to abort the run, edit his source file and re-run the model. To avoid this problem, GPSS10 provides the additional capability to set at any block or blocks, a "breakpoint". When a transaction reaches a block where a breakpoint has been set, a pause occurs automatically. Breakpoints may be set and removed by interactive commands issued during a pause.

The problem of how to generate a pause is two-dimensional, however. Often one wants to pause on a clock time criterion, rather than on a block basis.

During a pause, the user may decide that he would like another pause to occur at a particular time or after a certain additional time has elapsed. This capability is provided by the "STOP AT" and "STOP IN" commands.

When a transaction is moving through an undebugged or intricate protion of a model, it may be desirable to pause each time a transaction moves from one block to another. This feature is implemented by defining a mode of processor operation called "step mode", in which a pause occurs before each block move. This mode may be enabled or disabled during a pause by the "STEP ON" and "STEP OFF" commands.

Whenever a pause occurs, the system types an explanation of the condition causing the pause. This is an abbreviated trace message in the case of a breakpoint or PAUSE block or another appropriate message in the case of a keyboard interrupt or STOP interrupt. During a pause, the user may request other information about the current state of the model. Commands in this category include one to examine or change the next block number to be entered by any transaction, one to type the current value of any SNA and one to examine or change the operand values of the current block. In addition, the values of savevalues, parameters and matrix savevalue elements may be changed during a pause.

GPSS10 has enhanced trace capability. Three kinds of trace are provided, differing in amount of information produced and output device. A trace flag is associated with each transaction, as usual, but also with each block. When a transaction enters a trace-flagged block, that block move is traced just as if the transaction trace flag had been on. Both blocks and transactions may be created with their trace flags turned on. During a pause the user may turn on or off any trace flag in the model. Other features permit generation of a snapshot report for later examination and provide the ability to abort the run gracefully.

During a pause, the GPSS10 user may define a macro containing several interactive commands. These commands may be executed by giving the macro name itself as a command. Future plans call for associating a macro name with a PAUSE block, breakpoint or STOP request. This macro would be invoked automatically whenever the associated pause occured, permitting semi-automatic generation of trace output to suit a user's requirements.

The commands which have been described permit very efficient monitoring of a model. They allow a significant reduction in the time required to debug a large model and worthwhile savings can be realized even with a model of modest complexity.

## CONCLUSION

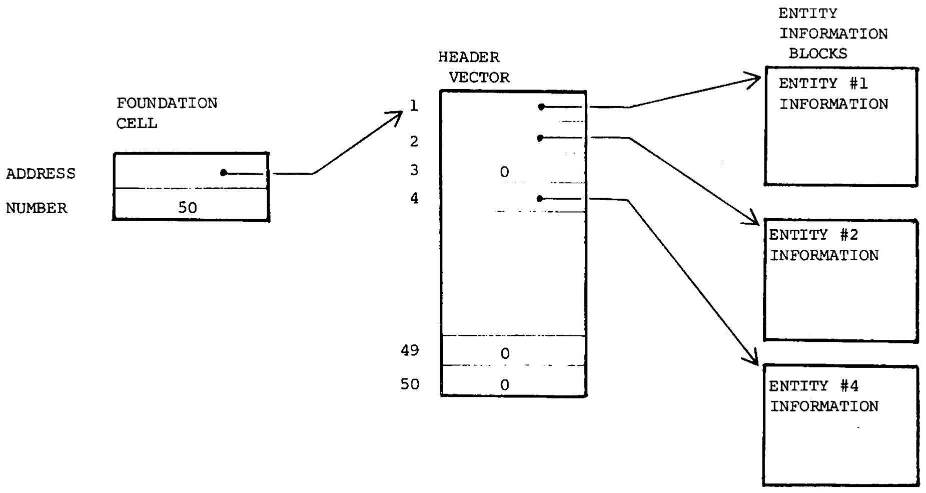
We have examined some internal features of GPSS10, an implementation of GPSS for the DECsystem10, showing the influence of storage and processor time considerations on the design of some internal structures. We have also explored the GPSS10 approach to interactive debugging and validation of models, noting the contrast that exists between it and the approach taken in other interactive GPSS languages and describing those features of GPSS10 which permit examination and modification of a model or a part of a model in greater or in lesser detail.

## REFERENCES

1. W.A. Wulf, et al., "BLISS: A Language for Systems Programming", CACM, Vol. 14, No. 12, 780-790, December, 1971.

2. "SIMULA 67 for the DECsystem10 - Functional Specification", 8-1 to 8-35, March, 1973.

3. James K. Mullin, "A Comparison of Two Dynamic Memory Allocators", Proceedings of the Canadian DECUS Symposium, (to appear).



<u>DATA STRUCTURE FOR AN ENTITY TYPE WITH 50 ENTITIES OF THAT TYPE ALLOCATED</u>

## SUMMARY OF GPSS10 INTERACTIVE COMMANDS

### ENTRY TO INTERACTIVE MODE

- -by keyboard interrupt
- -by a transaction reaching a conditional for unconditional PAUSE block
- -by a transaction reaching a breakpoint block
- -by clock condition due to previous STOP request

### OUTPUT REQUESTS

- -to display the current value of any SNA
- -to display the current value of the operands of the current block
- -to display the next block number of any active transaction
- -to generate a snapshot report for later examination

### CONTROL REQUESTS

- -to terminate the run and produce a standard GPSS report
- -to terminate the run by a fatal error, generating an error report
- -to terminate the pause and resume simulation
- -to enable or disable the step mode of processor operation

### MODIFICATION REQUESTS

- -to set or clear a block trace flag
- -to set or clear a breakpoint at a block
- -to change the value of a matrix element, savevalue or parameter
- -to change the value of an operand of the current block
- -to change the next block number of any active transaction
- -to set or clear any trace flag in the system

### MACRO RESULTS

- -to define a macro
- -to display the text of a macro
- -to delete a macro definition
- -to list the names of defined macros