# BUSINESS ORIENTED SIMULATION SYSTEM (B.O.S.S.):

# A SIMULATION LANGUAGE BASED ON COBOL

Andreas Philippakis
Arizona State University, Tempe, Arizona 85281

## Introduction

Simulation applications are often characterized by extensive effort over extended periods of time, involvement of a number of individuals, and a long use life marked by experimentation and modification.

These characteristics make it highly desirable for simulation models to be easily coded into computer programs and that the main logic of these program models be highly readable. Ease of coding is a requirement following from the fact that many simulation projects represent extensive modelling efforts and therefore provide potential for substantial effort reduction. The readability requirement follows from the fact that several people may be involved directly over the life of the project and communication must be facilitated. In addition, the readability requirement follows from the long life of simulation projects. Cryptic, terse codes serve expeditious needs in the short run for the analyst currently immersed in the project, but become stumbling blocks in the long run, both for the original author whose retension of nonsense syllable codes is limited and for subsequent users of the model who find learning the unnatural vocabulary a nuisance task.

But, in addition to those characteristics that dictate easy coding and readability, simulation projects are also characterized by the important need to be capable of handling efficiently vast numbers of numerical processes such as random number generation and statistical collection and analysis. This paper reports on an effort to develop a simulation language which facilitates the coding of discrete event simulation models so that they are highly readable for ease of initial and subsequent modelling but also computationally efficient for carrying out simulation experiments.

## Overview of B.O.S.S.

The B.O.S.S. language is designed for discrete event [2] digital simulations. It utilizes the concepts of "event", "attribute" and "file" in a fashion similar to GASP II [3], another discrete event simulation language*.

An overview of the language follows while a more extended description is given in the next section. The main characteristics of B.O.S.S. are:

The main program is written by the user in COBOL. To appease the reader who may be frightened by the specter of voluminous ENVIRONMENT and DATA DIVISION codes, it is added that only minor effort is required outside of the PROCEDURE DIVISION.

The user needs only knowledge of a very limited subset of COBOL since the nature of simulation tasks does not lend itself to the great majority of COBOL options.

Data handling is carried out by COBOL subprograms which, themselves, take full advantage of COBOL data processing and data structure capabilities.

Statistical data collection -- the heart of simulation models -- is performed automatically without the need for explicit instructions. The user simply specifies as input data, the variables and the options desired for statistical collection.

Event generation is a powerful but simplified process. An event may be characterized by multiple attributes whose values may be random variables.

Heavy emphasis is placed on formatted input record which, in the form of a readable questionnaire, requires the user to specify the basic characteristics of the model. This input record has the double effect of providing an explicit document for subsequent reference and relieving the user from a number of concerns during model logic development.

Output documentation is unusual. Instead of general labels, output is headed by labels supplied as data for each variable, file, and attribute.

Debugging aids are available which enable the user to trace the executed sequence of steps and aid the process of error detection.

The language is designed so that special purpose modules may be added to simplify coding requirements for models that fall into standardized categories, such as queueing, inventory, and networks.

## Elements of B.O.S.S.

Figure 1 shows a summary of the language elements. A brief description of each element follows:

1. A call to INPUT involves the reading in of the input data and model parameters specified in a questionnaire type input coding form. This input form forces considerable amount of planning on the user -- a requirement which should serve the useful purpose of aiding the complete conceptualization of the model prior to actual coding.

2. The EVENTS element is capable of generating future events or initiating the processing of the imminent events. An event can be a multi-attribute entity whose attribute values may be constants or random variables. Statistical accumulations can be performed on any desired attribute. One special event can be specified which causes intermittent output on a scheduled basis.

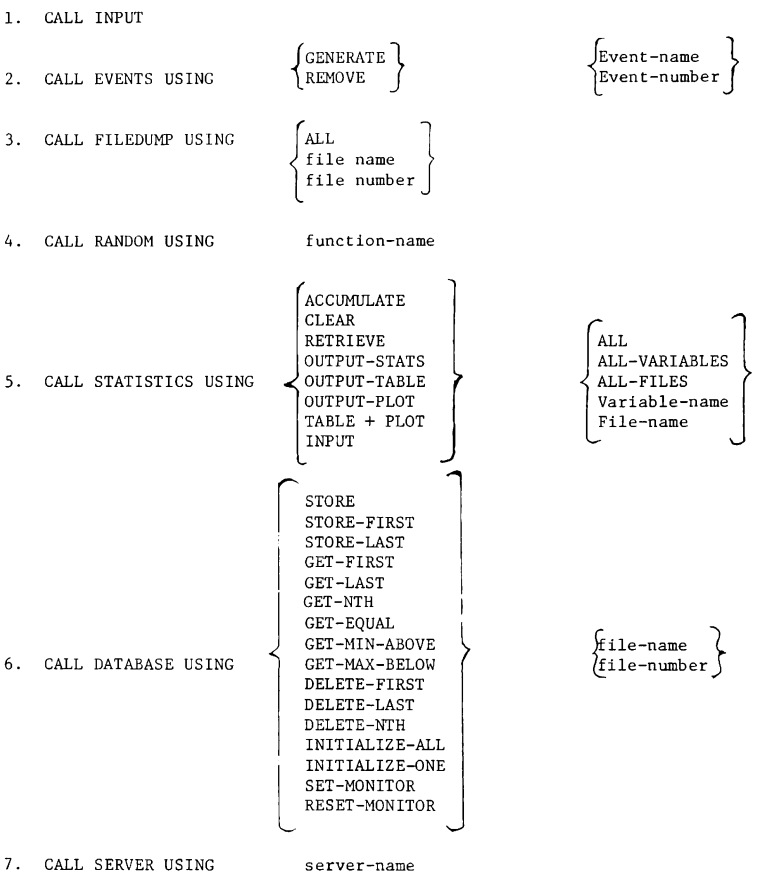
3. The FILEDUMP element outputs, in a well labeled format, file contents for analysis, documentation, or diagnostic purposes.

4. RANDOM generates a random variate from a function whose name specifies a distribution. There are three types of distributions. The first type includes standard probability distributions such as normal, uniform, exponential, etc. The second type is a user-defined table function, and the third is one providing for invariable recurrence.

5. STATISTICS is used mainly to generate statistical output for all variables, or for a selected variable. Output options include frequency tables and histogram plots.

*The author wishes to acknowledge the direct influence of GASP in the development of this language.

Figure 1 B.O.S.S. Language Elements




6. The DATABASE element provides the user with a powerful tool for handling collections of data in files. A number of options are available each specifying a particular function. The names of these functions are self-explanatory with the exception of a few. The INITIALIZE function sets file pointers so that files are logically empty. The SET-MONITOR and RESET-MONITOR functions are debugging aids which allow selective tracing of data flow during model execution.

7. The SERVER module exemplifies the capability of the language to allow the specification and use of special-purpose modules. The SERVER is designed for queueing models. The invocation of SERVER processes an arrival event by scheduling an end-of-service event if the specified server is free, or by storing the arriving entity into a specified queue. In addition, when an entity has been serviced by the specified server, its arrival to the next server, should there be one, is processed and, when available, the first entity in the queue is assigned for service to the current server.

A detailed look at the elements of the language would exceed the scope of this paper. The reader may, however, get a more concrete idea of the use of B.O.S.S. by turning to a simple example application which may help bring some of these concepts into focus.

## An Example Application

A simple inventory system is to be simulated. Starting with 100 units of inventory on hand the model proceeds through time incurring two main events, issues and receipts. A third event is scheduled every 200 time units to print intermediate results. The simulation terminates after 1000 time units.

An issue represents a withdrawal from inventory. Issues occur at time intervals characterized by a specified probability distribution (for example normal). The amount of each issue is a random variable with a specified probability distribution (for example uniform). When the amount of an issue exceeds the inventory on hand, a stock-out condition exists and a partial or complete order is lost.

We are interested in collecting statistics on two variables. The first one, INVENTORY-ON-HAND is time dependent. The second one, STOCKOUTS, accumulates data on the size of lost orders.

Figure 2 shows the DATA DIVISION entries which the user had to write for this application, and the basic event logic expressed in the PROCEDURE DIVISION of a program written to carry out this simulation. It will be noted that the language used allows a great deal of readability. It will also be noted that the number of characters per instruction is high, expecially when compared to a language like FORTRAN. It is the belief of this author that the cost of additional characters is inconsequential. The effort required in developing a simulation program is not related to the physical effort of writing instructions or the clerical task of keypunching. If longer instructions increase the visibility of program logic both at the time of initial development and debugging and at subsequent uses of the program it seems that longer instructions are a very small price, indeed.

## Experience with B.O.S.S.

Experience with the language has been limited. A number of example models have been coded including one which incorporates a FORTRAN search routine by Schmidt and Taylor [4] and an autoregressive scheme by Fishman [1]. An undergraduate simulation class has received a three-week instruction in the use of the language and as a result several students were able to code a complete problem. The main limitation up to this point has been the lack of any printed instructions on the use of the language. Efforts are under way to put together a user's manual.

The current version of BOSS has been implemented on a large-scale UNIVAC 1110 system. Transfer of the language to other systems should not entail more than routine conversion efforts.

## References


1. Fishman, George S., Concepts and Methods in Discrete Event Digital Simulation. John Wiley & Sons, New York, 1973.

2. Kivat, Philip J., Discrete Event Simulation: Modeling Concepts. The Rand Corporation, RM-5378-PR, Santa Monica, California, 1967.

3. Pritsker, A. Alan B. and Philip J. Kiviat, Simulation with GASP II. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.

4. Schmidt, J.W. and R.E. Taylor, Simulation and Analysis of Industrial Systems. Richard D. Irwin, Inc., Homewood, Ill., 1970.


Figure 2

```
01  QUANTITY                  USAGE COMP-1.

01  STATISTICAL-VARIABLES.
    02 INVENTORY-ON-HAND      USAGE COMP-1.
    02 STOCKOUTS              USAGE COMP-1.
    02 UNUSED                 USAGE COMP-1 OCCURS 38 TIMES.

PROCEDURE DIVISION.
MAIN-PART.
    CALL 'INPUT'.
    MOVE 100.0 TO INVENTORY-ON-HAND
    PERFORM SIMULATION-START THRU SIMULATION-END
       UNTIL CURRENT-TIME > 1000.0.
    CALL 'STATISTICS' USING TABLE-PLOT 'STOCKOUTS'.
    STOP RUN.
*
SIMULATION-START.
    CALL 'STATISTICS' USING  ACCUMULATE 'ALL      '.
    CALL 'EVENTS' USING REMOVE-EVENT 'NEXT-EVENT'
    MOVE FILE-ATTRIBUTE (4) TO QUANTITY.
    GO TO ISSUE, RECEIPT, RESULTS
         DEPENDING ON EVENT-CODE.
*
ISSUE.
    IF INVENTORY-ON-HAND IS LESS THAN QUANTITY
       SUBTRACT INVENTORY-ON-HAND FROM QUANTITY GIVING STOCKOUTS
       MOVE ZERO TO INVENTORY-ON-HAND
    ELSE
       SUBTRACT QUANTITY FROM INVENTORY-ON-HAND.
    CALL 'EVENTS' USING CREATE-EVENT 'NEXT-ISSUE'.
    GO TO SIMULATION-END.
*
RECEIPT.
    ADD QUANTITY TO INVENTORY-ON-HAND
    CALL 'EVENTS' USING CREATE-EVENT 'NEXT-RECEIPT'.
    GO TO SIMULATION-END.
*
RESULTS.
    CALL 'EVENTS' USING CREATE-EVENT 'NEXT-RESULTS'.
    CALL 'STATISTICS' USING OUTPUT-STATS 'ALL      '.
    CALL 'FILEDUMP' USING 'ALL      '.
SIMULATION-END.
    EXIT.
```