1981 Winter Simulation Conference Proceedings
T.I. Ören, C.M. Delfosse, C.M. Shub (Eds.)

IMBEDDING GPSS IN A GENERAL PURPOSE PROGRAMMING LANGUAGE

Jerrold Rubin

IBM Corporation
P. O. Box 12195
Research Triangle Park, N. C. 27709

ABSTRACT

GPSS has proven to be an excellent simulation language, but was not designed to perform the logical and computational tasks of a programming language. Strategies for improvement can take one of two paths: building adequate analytic constructs into the existing GPSS language, or building GPSS-like constructs into an existing general purpose programming language. The second choice can be made easier by use of a language-building utility program to translate simulation constructs into the selected programming language. It is also simplified by the fact that some GPSS statements (such as input-output and control) may be dropped, since any good programming language will provide native facilities for these functions. Such a fusion between the constructs of a good simulation language, and those of a good programming language, provides a far more flexible system than either alone. This paper discusses PL/I GPSS, an implementation of GPSS in a PL/I environment which uses PL/I variables as transaction variables and permits use of general PL/I expressions for most GPSS statement parameters.

1. INTRODUCTION

The General Purpose Simulation System (GPSS) has become one of the most widely used simulation languages since its introduction in the 1960's. Its greatest strength is that it enables the user to easily simulate situations in which resources are utilized for varying periods of time. However, complex simulations often require mixing powerful simulation commands with the flexible type of computational and logical capabilities provided by a good programming language. The IBM product GPSS V [1] does have the ability to exit to PL/I or FORTRAN in order to gain this additional flexibility, but there is little question that a native capability of this type is highly desirable. There appear to be two courses open to achieving this: either additional function must be built into GPSS, or GPSS capabilities must be built into a language which already has the desired function - preferably one of the major programming languages. We describe here an imbedding of GPSS in PL/I which is now available as an IBM product [2].

2. STRUCTURE OF PL/I GPSS

In principle, each GPSS function could have been provided as a PL/I preprocessor macro. However, it proved more efficient to utilize a program known as the PL/I Language Construction Aid [3]. This provided a capability for easily defining new commands within PL/I, with the added advantage that command verbs (SEIZE, ENTER, etc.) need not become reserved symbols. Using this product, a system has been built which effectively combines GPSS and PL/I.

In order to develop the system quickly, a second more primitive and more flexible set of simulation commands have been implemented, upon which the GPSS commands are based. This subsystem of PL/I GPSS is called SIMPL/I X, a version of the SIMPL/I programming language [4], and provides certain very basic simulation facilities:

81CH1709-5/81/0000-0033$00.75

a. to define processes, each with its own data

b. to provide reference to a process by a pointer to the data associated with it

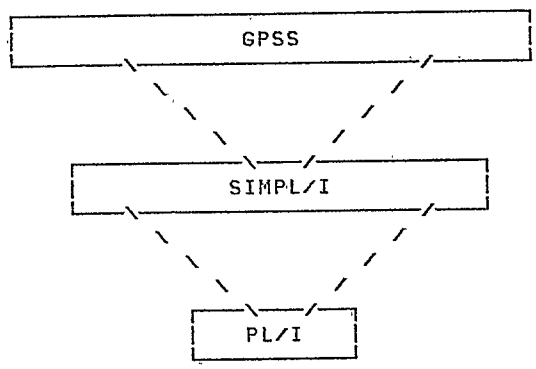
c. to delay a process for a stated time

d. to delay a process until notified by another process to continue

e. to construct and manipulate lists of processes

The more powerful GPSS commands can then be automatically translated into the more fundamental SIMPL/I commands and thence to PL/I. The Language Construction Aid provides this multiple translation facility transparently to the user.

The simulation designer is free to use GPSS commands, SIMPL/I commands, or PL/I statements, freely intermixed. Thus, he has available a three level hierarchy of commands, each built upon the levels below.



Such a system provides the following advantages:

a. efficiency of computation through use of compiled code

b. data base access via PL/I

c. a finer degree of control over the simulation than GPSS V provides

d. more flexible data definitions - transaction variables are PL/I variables, and most command parameters are PL/I expressions

e. extensibility - the Language Construction Aid permits definition of new application-oriented commands

There are also some disadvantages in this approach:

a. Unlike GPPS V, a compilation is required

b. it is not fully compatible with GPSS V (though easy to translate) in part due to the syntax requirements imposed by the Language Construction Aid, in part by deliberate choice.

3. PL/I GPSS Syntax

The discussion below assumes some knowledge of the semantics of GPSS commands and keywords. A PL/I-GPSS command normally consists of a command name, with some parameters in parentheses, followed by one or more keywords, each with optional parameters, and finally a semicolon -

i.e., GENERATE(10) PRIORITY(P);

Generally, positional parameters are utilized for the most commonly used parameters, to minimize writing, and keywords for the more rarely used parameters, as an aid to memory.

Value Macros

There are also certain "value" macros which return a value. Distribution functions and GPSS run statistics fall into this category.

```
X = $NORM(1,mean);
```

will sample from a normal distribution using random number stream 1.

Interspersing PL/I

PL/I statements may be freely intermixed with GPSS commands:

```
X = X+1;
IF X>LIMIT THEN SEIZE(F);
```

In addition, most macro parameters may be PL/I expressions:

```
GENERATE(GENFCN, ...);
        -
        -
        -
GENFCN:PROC RETURNS(DEC FLOAT);
```

Symbols for GPSS Entities

GPSS entity symbols (representing storages, facilities, etc.) may be assigned values by the system, or the user may choose to control the assignment. In the latter case, an * precedes the symbol:

System assignment:

```
no declaration needed
ENTER(S);
```

User assignment:

```
DCL N FIXED BIN INIT(10);
   -
   -
   -
ENTER(*N);
```

The PL/I-GPSS system will generate any needed declarations for system controlled entity symbols. System controlled entity symbols are assigned values starting with one higher than the maximum number of user controlled symbols. These maximums (50 each by default) may be specified on the SIMULATE statement (up to 16383 each).

GPSS Transactions and Transaction Data

A PL/I GPSS transaction type is defined by a TRANSACTION statement, followed by a PL/I substructure containing the variables associated with each transaction of that type.

```
TRANSACTION(T),
   2 VAR1(10) FIXED BIN,
   2 (VAR2,VAR3) DEC FLOAT,
   2 VAR4 BIT(5);
```

The above describes a transaction type named T with ten fixed point variables, two floating point variables, and one five bit string.

4. ORGANIZATION OF A PL/I-GPSS SIMULATION AND TRANSACTION FLOW

A PL/I GPSS program is organized into separately compatible sections, each headed by a SIMULATE or SECTION statement and ended by a GPSSEND statement. The initial section, at which the simulation is entered, is the only one headed by a SIMULATE statement. Normally, it contains control statements like START, RESET, etc., but may also contain portions of the simulation proper. Unlike GPSS V, control statements are executable. In fact, a control transaction is created by the system to execute the control section.

```
SIMULATE;
ACTIVATE(SECT1,SECT2,SECT3);
START(10);
GPSSEND;
```

The control transaction must eventually enter either a TERMINATE or a GPSSEND command, either of which will automatically terminate it.

5. EXAMPLE OF A PL/I GPSS SIMULATION

The XYZ company manufactures two items: widgets and wodgets. Widgets can be produced on any of three machines, but wodgets only on the last of these. The first available machine is used. Out of a total of five employees, it takes three individuals to supervise the manufacture of a widget, only one for a wodget. Once a machine is assigned to a widget or a wodget, it is reserved until enough men become available.

Twenty percent of widgets take 10 to 20 minutes, 40 percent take 15 to 25 minutes, and 40 percent take 30 to 40 minutes (uniformly distributed), and every wodget takes just 15 minutes to manufacture. Orders for widgets and wodgets arrive on the average every ten minutes (exponentially distributed).

By varying the number of men or machines, we could arrive at an optimum capital investment and labor situation which avoids untoward production delay.

The model might be coded as follows:

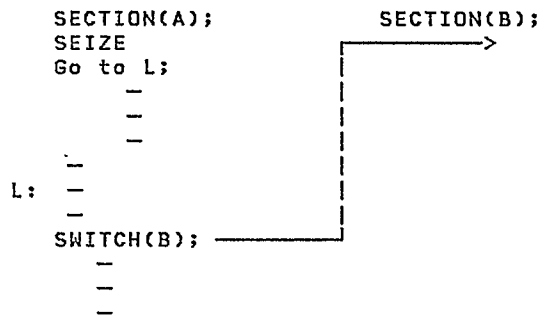
```
                                        POINTS OF NOTE
   SIMULATE STATS(FW,SW);               /*EXECUTABLE CONTROL SECTION          */
                                        /*AUTOMATICALLY CALCULATE             */
                                        /*    WAITING QUEUE STATISTICS        */
   STORAGE(MEN,5);                      /*STORAGE DYNAMICALLY DEFINED         */
   ACTIVATE(WIDGET,WODGET);             /*PREPARE THE GENERATES FOR           */
                                        /*    THESE SECTIONS                  */
   START(100) NP;
   RESET;
   START(1000);
   GPSSEND;

* PROCESS;

   SECTION(WIDGET);
   TRANSACTION(WIDGET),                 /*TRANSACTION TYPE DEFINITION         */
   2 MACHINE FIXED BIN;
   MIX(TIME,
            .2,$RANDFS(1,10,20),        /*DEFINE MIXTURE OF                   */
            .4,$RANDFS(1,15,25),        /*    DISTRIBUTION USING              */
            .4,$RANDFS(1,30,40));       /*    RANDOM NUMBER STREAM 1          */
   GENERATE($NEGEXP(1,10));
   GATEANY(BEGIN);                      /*NEW STATEMENT - GATEANY             */
   CONDIT($CS(MACH1),A);                /*TEST IF ANY MACHINE                 */
   CONDIT($CS(MACH2),B);                /*    CAN BE SEIZED                   */
   CONDIT($CS(MACH3),C);
   GATEANY(END);
   A:MACHINE=MACH1;                     /*'MACHINE' IS USER CONTROLLED        */
        GO TO D;
   B:MACHINE=MACH2;                     /*MACH1, MACH2, MACH3, MEN            */
        GO TO D;                        /*    ARE SYSTEM CONTROLLED           */
   C:MACHINE=MACH3;                     /*NOTE PL/I ASSIGNMENT,BRANCHING,LABELS*/
   D:SEIZE(*MACHINE);                   /*NO QUEUE STATEMENT NEEDED           */
   ENTER(MEN,3);
   ADVANCE(TIME);                       /*USE MIX OF DISTRIBUTIONS            */
   LEAVE(MEN,3);
   RELEASE(*MACHINE);                   /*USE OF '*' FOR USER CONTROLLED SYMBOL*/
   TERMINATE(1);
   GPSSEND;

* PROCESS;

   SECTION(WODGET);
   GENERATE($NEGEXP(1,10));             /*NO DCL NEEDED FOR SYSTEM            */
   SEIZE(MACH3);                        /*    CONTROLLED SYMBOLS, EVEN        */
   ENTER(MEN,1);                        /*    THOUGH USED IN SEPARATE SECTIONS*/
   ADVANCE(15);
   LEAVE(MEN,1);
   RELEASE(MACH3);
   TERMINATE(1);
   GPSSEND;
```

The above could have been combined into one section, but for purposes of illustration they have been coded as three sections, to be compiled separately.

It is possible to avoid using any PL/I code, but the example, as coded, illustrates the three most common uses of PL/I code: PL/I labels, assignment, and alteration of flow.

6. GENERATING TRANSACTIONS

The ACTIVATE command will cause GENERATE commands in referenced sections to start generating transactions. Transactions flow through GPSS commands and interspersed SIMPL/I or PL/I code until a branch or a TERMINATE statement is encountered. Branching within a section is effected by the PL/I GOTO statement. A transaction may be switched to a position within another section by a SWITCH command.



The sectionalizing of a PL/I, GPSS program is useful if a model grows too large for quick completion, or if several individuals work on different components. The scope of any PL/I variables used can be kept internal to a section in order to minimize conflicts between sections. Once a section is activated, each GENERATE statement in it will produce transactions at specified intervals until one of three conditions is satisfied:

a. the number of terminations specified on the START statement has been reached

b. the control transaction has terminated (which always ends the model)

c. a condition of the GENERATE statement is satisfied:

GENERATE(...) UNTIL(condition);

Sections can be reactivated after they have stopped generating transactions or even while they are still generating them. In either case a new stream of transactions will start.

Using SIMPL/IX it is also possible to create a transaction in one section which will flow through a different section:

STARTUP(T);

where, using the default options, a transaction of type T will flow through the section named T.

7. INTER TRANSACTION COMMUNICATION

In the SIMPL/IX sublanguage, upon which PL/I GPSS is constructed, a process (including a GPSS transaction) is identified by a PL/I pointer to its data structure. The process currently active is identified by the pointer CURRENT. A GPSS programmer may utilize this, and save CURRENT, perhaps on a list, with the effect that one transaction can reference or update the data structures of another. Such communication between transactions provides a considerable expansion in basic simulation capability.

In addition SIMPLI/X provides the ability to control the behavior of specific transactions. For example:

HOLD P;

will cause transaction P to wait (the waiting time not to be counted against the interval specified in an ADVANCE block) and

NOTIFY P;

will release the HOLD , and permit P to continue (possibly to continue taking time).

There are a variety of other statements which permit transactions to affect one another.

8. OTHER USES OF NATIVE PL/I and SIMPL/I

In addition to the uses illustrated above, native PL/I can be used for input-output. In particular, by utilizing system functions which return statistics for the run, the user may provide his own formatting for the output of a run, if he does not wish to use the default printout (which is similar to that provided by GPSS V).

A list processing capability is provided by the SIMPL/IX subsystem. This includes the ability to dynamically create lists, insert and delete transactions, and automatically maintain list statistics. The list processing facility replaces the GPSS V "group" facility, including such statements as JOIN, ALTER, SCAN, etc. Since the pointers to transaction data are accessible, one transaction may scan a SIMPL/IX list of others, and alter its behavior according to the state of any of the scanned transactions.

SIMPL/IX includes a variety of primitive simulation functions. In addition to the HOLD and NOTIFY statements already mentioned, transactions can be started and terminated (STARTUP, TERMIN), be made to take time (TAKE), or scheduled for a particular time (SCHEDULE). These commands are individually less powerful than the

more complex GPSS commands, but because they represent primitive simulation functions, they can be combined with a greater degree of flexibility. To take one instance, there is no very simple way in GPSS V to control transactions entering a storage, if the logic of the entry test differs from that assumed by GPSS V. For example, if the remaining storage capacity is 5, and transaction A tries to enter with 7 units, it will wait. If transaction B tries to enter with 3 units, it will succeed. If this continues to happen, transaction A can be indefinitely frozen out. This may or may not represent the real-world situation.

By using an explicit SIMPL/I list to enqueue transactions to the storage, and a manager of this list, one can easily tailor the logical requirements for entry into a storage.

```
STARTUP MANAGER SET(MANAGER_PTR);
  -
  -
  -
UNITS = ...
INSERT CURRENT INTO SLIST;
NOTIFY MANAGER_PTR;
HOLD;
  -
  -
  -
LEAVE(S,UNITS);
NOTIFY STORAGE_MGR;
```

```
SECTION(MANAGER);
M:DO WHILE (¬EMPTY(SLIST)&
   FIRST(SLIST)->UNITS<=$R(S));

   ENTER(S,FIRST(SLIST)->UNITS);
   NOTIFY FIRST(SLIST);
   REMOVE FIRST FROM SLIST;
END;
HOLD;
GO TO M;
GPSSEND;
```

The above logic would assure that transactions enter storages in the order in which they attempted to do so.

While it is more work to build this function in SIMPL/IX, more precision can be obtained. One need not make assumptions about the way resources are to be utilized, which are sometimes implicit in GPSS V commands.

9. RANDOM NUMBER GENERATORS AND DISTRIBUTION FUNCTIONS

System functions are included to generate random numbers and to sample from a variety of distributions. The following distributions are included:

Beta	Negative Binomial
Binomial	Negative Exponential
Erlang	Normal
Gamma	Poisson
Geometric	Uniform
Hypergeometric	User Defined (continuous)
Lognormal	User Defined (discrete)
	Weibull

These generally provide more accuracy than tabular approximations, although tabular distributions can also be used. Twenty random number streams are available with twelve unique seeds. (The seeds can be changed by the user.) Rerunning with different random number streams can illustrate the variability of the results of a simulation.

A histogram output capability, as well as a two dimensional plot rountine are also provided.

10. EXTENSIONS TO THE GPSS LANGUAGE

Although the main advantage of PL/I-GPSS is the ability to use PL/I directly within the simulation, certain language extensions to GPSS have been implemented. Some of these are listed below:

GATEANY	hold the transaction until any one of a set of specified conditions has been satisfied.
GATEALL	hold the transaction until all of a set of specified conditions have been satisfied.
GENERATE UNTIL	the UNTIL condition limits the generation, which can be reactivated by an ACTIVATE statement.
PRINT	provides broad control over output.

Assembly Groups	all transactions start in assembly group zero, so that transactions may be assembled without having gone through a SPLIT block.
Dynamic Storage Definition	permits easy modeling of replenishible resources.

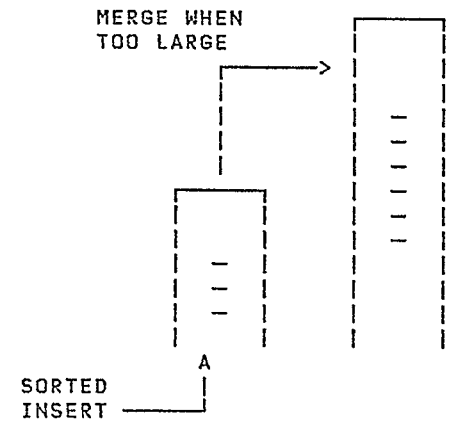
11. DEFINING NEW LANGUAGE STATEMENTS

By utilizing the Language Construction Aid, it is possible to define new commands, either general purpose simulation commands or commands tailored to a particular application. This is done by providing a "macro expansion" routine coded in PL/I. The string returned by this routine may include PL/I statements as well as GPSS or SIMPL/IX statements.

In another paper at this conference [5], Metz describes the use of this facility for creating a simulation package for a point-of-sale system, including an application-oriented input language.

12. CALENDAR MANAGEMENT

GPSS V manages transactions by utilizing a current events list for transactions which, if not blocked, could in principle execute at the current instant of time, and a future events list for transactions at ADVANCE blocks. By contrast, in PL/I GPSS all transactions are either on a calendar (scheduled to be invoked at a known clock value) or held on lists, to be scheduled onto the calendar when particular conditions are satisfied. The calendar itself is kept sorted, in order of clock time and transaction priority. It is maintained as two lists, one short and one long. Insertions into the calendar are always made into the short list. Whenever the size of the short list[1] exceeds the square root of the size of the long list, the short list is merged into the long list. To determine the next transaction to be activated, the first item of each list is examined and the earlier scheduled transaction chosen.



INSERT TIME = SQRT(CALENDAR SIZE)

For this method, bookkeeping time for a single transaction can be shown to be proportional to the square root of the average number of entries on the calendar. Its simplicity recommends it as an alternative to a binary tree method, for example. It is capable of handling thousands of transactions simultaneously on a calendar (and thousands more held on lists) without undue bookkeeping time. Of course, for most typical simulations, perhaps a few dozen transactions will exist simultaneously, and the benefits of any calendar management scheme over any other become marginal. Nevertheless, the capability for managing massive numbers of simultaneous transactions without risking thrashing is useful for certain types of applications.

---

[1] Actually, for optimality, the merge test is on a function of the square root of size, depending on inner loop path lengths for merge and insert.

REFERENCES

[1] IBM Corporation (1973), General Purpose Simulation System V - OS Operations Manual (Program Number 5734-X52), Publication SH20-0867-3, Data Processing Division, White Plains, New York.

[2] IBM Corporation (1981), PL/I General Purpose Simulation System Program Description/ Operations Manual (Program Number: 5796-PNN), Publication SH20-6181-0, Data Processing Division, White Plains, New York.

[3] IBM Corporation (1980), PL/I Language Construction Preprocessor Description/ Operations Manual (Program Number: 5796-PLL) , Publication SH20-2164-0, Data Processing Division, White Plains, New York.

[4] IBM Corporation (1972), SIMPL/1 (Simulation Language Based on PL/I): Program Reference Manual, Publication SH19-5060-0, Data Processing Division, White Plains, New York.

[5] Metz, W. (1981) "Discrete Event Simulation Using PL/I Based General and Special Purpose Simulation Languages," Proceedings of 1981 Winter Simulation Conference, Atlanta, Georgia, December 1981.