DISCRETE EVENT SIMULATION USING PL/I BASED GENERAL AND SPECIAL PURPOSE SIMULATION LANGUAGES

Walter C. Metz
Systems Communications Division
International Business Machines Corporation
G39/B602
P.O. Box 12195
Research Triangle Park, N. C. 27709

ABSTRACT: This paper describes the architecture and language features of a simulation model which was developed using a new IBM discrete event simulation package based on PL/I. The package contains implementations of both the GPSS and SIMPL/I simulation languages, and in addition provides the capability for a model developer to create special purpose simulation languages tailored to his unique simulation application. The model described in this paper simulates a retail or supermarket store point-of-sale communication system. The model was written using both GPSS and SIMPL/I commands, but with a limited subset of the available commands in each language. Three special purpose languages were developed so that a model user can easily create a version of the model that simulates a particular store system. The model description illustrates the various language alternatives available with this new simulation package.

1. INTRODUCTION

A new discrete event simulation package consists of two IBM developed programs which are available as IBM products: (1) PL/I General Purpose Simulation System (IBM 81), and (2) PL/I Language Construction Preprocessor (IBM 79). Together, the two programs provide versions of the GPSS and SIMPL/I simulation languages, allowing models to be written with simulation commands interspersed with PL/I statements. The version of GPSS is called PL/I GPSS, and the version of SIMPL/I is called SIMPL/I X. The two languages are implemented using the PL/I Language Construction Preprocessor which accepts as input a program consisting of SIMPL/I and GPSS commands interspersed with PL/I statements and produces as output a standard PL/I program acceptable to the IBM PL/I Optimizing Compiler. Procedures have been established for installing the programs on both the IBM MVS and CMS operating systems. See the conference paper by Jerrold Rubin for a general presentation of the overall features of PL/I GPSS and SIMPL/I X.

The new package has a number of advantages over the earlier IBM program products, GPSS V (IBM 77) and SIMPL/I (IBM 71). It offers the model developer a wider range of simulation language features and program design alternatives. He has a choice between two "world views", the "transaction flow orientation" of GPSS or the "process orientation" of SIMPL/I (Shannon 75). He may combine the two orientations, since with certain restrictions, a simulation program may be written using both GPSS and SIMPL/I commands. This should not be surprising, since the two languages have sometimes been categorized together as using the "process interaction" approach (Fishman 78).

One of the most useful features of the package is the language expansion capability provided by the PL/I Language Construction Preprocessor. Special purpose commands may be added to the preprocessor to supplement the general purpose GPSS and SIMPL/I commands. For each special purpose command, as with each SIMPL/I and GPSS command, the preprocessor must contain a command expansion routine whose function is to supply a string of code to replace the command. The generated code, which is conditional upon the keyword parameters in the command, may include any SIMPL/I, GPSS, or special purpose command, since the preprocessor optionally rescans each command expansion for further commands, which are expanded in turn. By including commands in command expansions, a hierarchy of simulation languages may be developed. Fig. 1 illustrates the relationship between these language levels and how one level builds upon another.

Level	Language	Written in terms of
4	Special Purpose	GPSS,SIMPL/I,PL/I, Special Purpose
3	GPSS	SIMPL/I,PL/I
2	SIMPL/I	PL/I
1	PL/I	----

Fig. 1 Hierarchy of Language Levels

The PL/I Language Construction Preprocessor is more efficient and versatile than the built-in preprocessor of the PL/I compiler. Full use of the PL/I language is allowed in the command expansion routines as opposed to limited use permitted by the built-in preprocessor. Efficiency is achieved by pre-compiling the command expansion routines and avoiding any extra processing for an unreferenced command. To simplify the writing of the command expansion routines, subroutines are provided in the preprocessor for decoding the commands.

The simulation package has been used at IBM to develop a model which simulates a retail or supermarket store point-of-sale communication system. The model, called SIMPOS, is used to evaluate system performance in terms of response times and throughput capacity. It contains both GPSS and SIMPL/I commmands, but only a limited subset of the available GPSS and SIMPL/I commands are used. Three special purpose languages have been developed for the model, the first to model store traffic, the second to model IBM microcode, and the third to model IBM and customer developed application programs. Development of these special purpose languages has made the model much easier to develop, maintain, and use.

This paper will first briefly describe the architecture of SIMPOS and important facts about the model such as the reasons for choosing PL/I GPSS as the base simulation language, the GPSS and SIMPL/I commands used in the model, the reasons for developing the three special purpose languages, and the model efficiency in terms of preprocessor, compile, and execution times. Then the features of the three special purpose languages will be presented with examples of each.

2. SIMPOS ARCHITECTURE

An IBM retail or supermarket store communication system contains a variable number of point-of-sale terminals which are connected to a store controller via a loop communication line. The controller with a disk storage device handles price lookup requests from the terminals, collects item movement data, and maintains accounting totals. The controller also communicates with a "host" computer via a switched line. The terminal and controller point-of-sale applications are programmable in a language similar to S/370 Assembler Language and executable by an interpreter microcode program.

The architecture of SIMPOS is illustrated in Fig. 2. A user of the model modifies or writes three simulation programs: (1) a traffic program, (2) a terminal application program, and (3) a controller application program. The traffic simulation language is used to write the traffic program, and the application simulation language is used to write both the terminal and controller application programs. A traffic program typically contains between 40 and 60 lines of code, a controller application program between 500 and 1000 lines of code, and a terminal application program between 50 and 150 lines of code. These numbers include simulation commands and PL/I statements, but exclude comments.

The linkage between the traffic program and the two application programs is a list of variable declarations called the message variable list, which is included in each program by the preprocessor in response to a %INCLUDE statement. During model execution, the traffic model generates message traffic and values for the variables in the message variable list according to specified probability distributions. These variables determine the functions to be performed in the terminal and controller application models. A message variable list typically contains between 5 and 25 variables.

The model user must also provide an input parameter file, which is read at the beginning of model execution. Its implementation permits many model runs to be made without recompiling model components. The input parameter file has the format of a macro language with statements such as NUMTERMS=12, RUNTIME=3.5, and TRACE=YES. It contains configuration, system generation, and application program parameters. In addition, it contains simulation parameters such as simulation run time, requested simulation output statistics, and indicators which specify the tracing of particular model components.

The remaining components of the model are not modified by the typical model user, but only by a model developer and a model user evaluating hardware and microcode design alternatives. Parameters for various hardware and microcode components can be set in an input parameter file. Examples are processor instruction execution rates and microcode program priorities. As is shown in Fig. 2, there are ten model components simulating microcode and hardware. A single interface exists between the controller interpreter model and the remaining controller microcode models. Since the terminals do not have multiple tasks contending for resources, the terminals are simulated in much less detail than the controller and require just the single terminal interpreter model component. These microcode and hardware simulation programs typically contain between 200 and 500 lines of code, including simulation commands and PL/I statements, but excluding comments. The simulation commands are primarily microcode language commands, except in the microcode control program model component, which contains GPSS and SIMPL/I commands in addition to the microcode language commands.

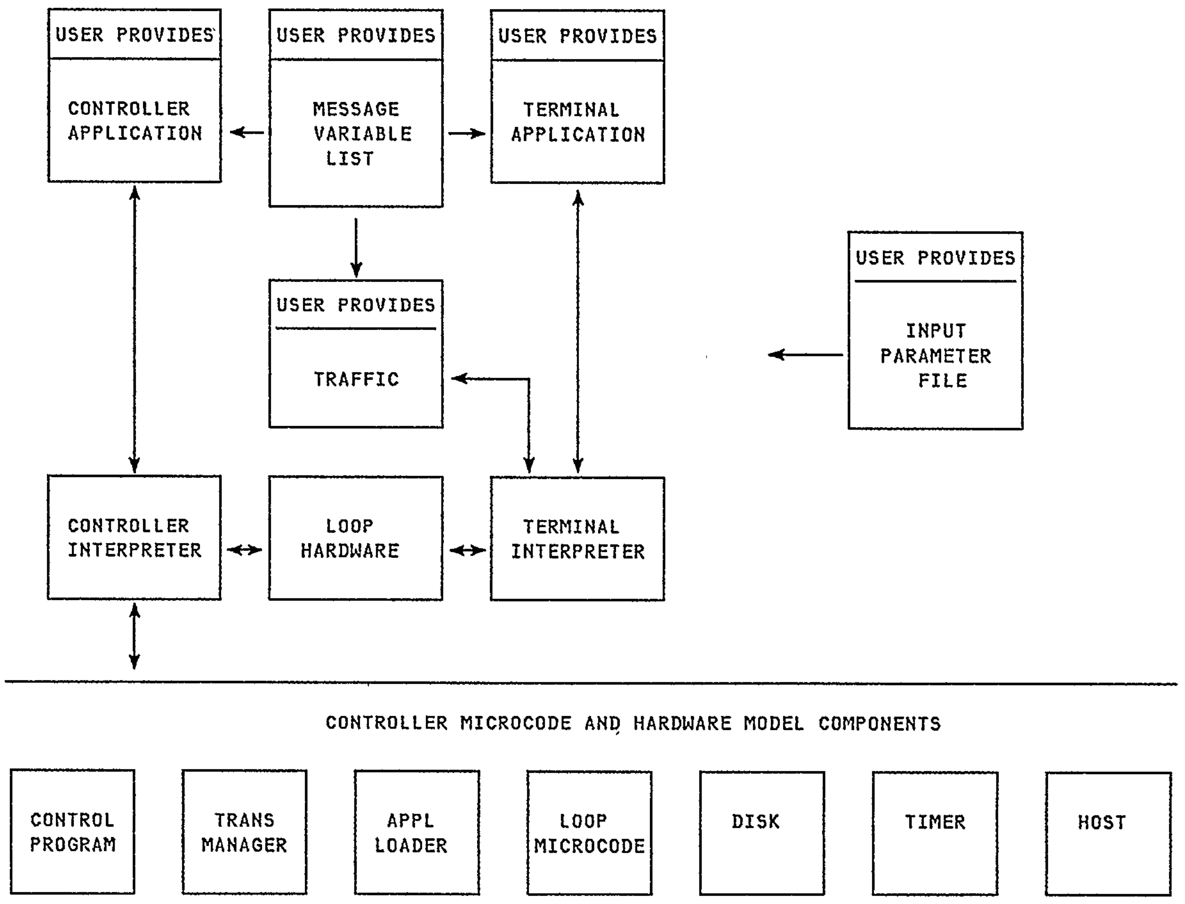

Fig. 2 SIMPOS Architecture

The programs in the model are separately compiled. Communication between the programs is with PL/I external variables which are included in each program by the preprocessor.

3. SIMPOS DEVELOPMENT

3.1 Choice of the Base Simulation Language

The IBM group which developed SIMPOS has traditionally used GPSS V to model point-of-sale systems. However, PL/I GPSS was chosen as the base language for SIMPOS because it has numerous advantages over GPSS V. These include the capabilities provided by incorporating GPSS in a full programming language:

- Data base access for input parameter files,
- Better computational facilities,
- Better editing facilities for writing output reports,
- Structured control statements such as IF-THEN-ELSE, DO-UNTIL, DO-WHILE, and SELECT-WHEN-OTHERWISE, and
- Better documentation resulting from longer identifier names (up to 31 characters for local and 7 for global identifier names, compared to 5 for all GPSS V identifiers).

In addition to providing the full spectrum of PL/I language facilities, PL/I GPSS has other advantages:

- Models can be divided into separately compiled sections, making it easier for several model developers to simultaneously write, compile, and execute different versions of the model. In contrast to GPSS V, where all entities are global, separately compiled PL/I GPSS sections result in most variables being local.
- SIMPL/I commands can be mixed with GPSS commands since the GPSS transaction is essentially a SIMPL/I process.
- The preprocessor permits the incorporation of additional simulation commands.
- PL/I GPSS has an extensive library of functions for generating random variates from the common discrete and continuous probability distributions.

These advantages were thought to far outweigh the disadvantage of PL/I GPSS of requiring preprocessing and compilation. The preprocessor and compile times for SIMPOS will be quantified later in the paper.

A few small models with known analytic solutions, such as a preemptive-resume M/M/1 queue with five priorities, were written in both PL/I GPSS and GPSS V to compare the results and the execution time. The two results were statistically equivalent and in agreement with the analytic results, and their execution times were roughly equivalent. It was anticipated that on a large simulation model with logic written in PL/I structured control statements, a PL/I GPSS model would execute considerably faster than the same model written in GPSS V. This proved to be the case and will be quantified later in the paper.

The basic time unit in the model was chosen to be one microsecond, which is in the same order of magnitude as the execution time of one microcode instruction. Since the PL/I GPSS clock is maintained as a 31-bit integer, the maximum time for a model run is approximately 36 minutes. This run time maximum is sufficient since the typical model run time is between 5 and 10 minutes.

3.2 SIMPL/I and GPSS Commands Used in the Model

Initially, the model was designed to be written with only GPSS simulation commands since the model developers had experience using GPSS V. Exclusive use of GPSS requires only a limited knowledge of PL/I and does not require the use of PL/I pointer variables. By contrast, SIMPL/I, a more primitive language than GPSS, gives the model developer more control over the simulation, but it requires a more detailed knowledge of PL/I and an extensive use of PL/I pointer variables. PL/I pointer variables, when used improperly, can lead to errors which cannot be detected at compile time but can be detected and corrected with difficulty at model execution time. (In PL/I flexibility has been achieved at the expense of limited compile-time error detection. The advantages of "strong typing", which insures better compile-time error detection, has been emphasized by proponents of Pascal based simulation languages (Bryant 80)).

Although a GPSS program can be written without pointer variables, the use of a pointer variable to reference a GPSS transaction permits a GPSS transaction to reference or update another GPSS transaction. This inter-transaction communication, a feature not provided in GPSS V, provides additional flexibility in model design.

In the development of SIMPOS, the choice was made to decrease execution time and to simplify the model design by using pointer variables and by converting several functions initially written in GPSS into SIMPL/I. In many situations, the GPSS commands contained more function than was required. For example, the GPSS ADVANCE command was replaced with the SIMPL/I TAKE command whenever time was not being taken on a resource. Also, GPSS LINK and UNLINK commands were sometimes replaced with SIMPL/I HOLD and NOTIFY commands, and other times with SIMPL/I INSERT and REMOVE commands. The result is that the model uses only a small subset of the available GPSS commands and a likewise a small subset of the available SIMPL/I commands.

A GPSS transaction is used to represent each task in the system. Each active task contains a PL/I pointer to a GPSS transaction which represents an operator entered message. These message transactions contain the message variables which determine the execution path in the terminal and controller application programs.

With minor exceptions the GPSS commands and SIMPL/I commands are used primarily in the microcode control program model. The SIMPL/I HOLD and NOTIFY commands are used to deactivate and activate the tasks respectively. The SIMPL/I INSERT and REMOVE commands are used to enqueue and dequeue tasks to the priority level queues. The GPSS PREEMPT, RETURN, and PRIORITY commands are used to model the preemptive-resume queueing discipline which is required for simulating the contention for controller processor cycles. The GPSS LOGIC command is used to override normal execution priorities in the control program.

3.3 Development of Special Purpose Languages

The three special purpose simulation languages were developed for several reasons:

- Their implementation reduces the source code and provides for good documentation in the model components, making the model easier to develop, maintain, and use.
- The model developer who writes microcode model components and the model user who writes application model components are concerned only with the system being modeled and relieved of simulation details such as gathering simulation statistics.
- Since much of the model structure is imbedded in these special purpose languages, improvements in model design can be made by changing only the command expansion routines. In these cases, the model components remain unchanged and require only a preprocessor and compile step.
- Each language can be made consistent with a particular modeling philosophy. Because of the static nature of the microcode, the microcode language was designed so that the microcode could be modeled in an abstract manner. This improves execution efficiency. By contrast, application programs may be frequently modified or rewritten. Thus, the application language was designed so that the applications could be modeled in an emulative manner, making changes to the model very easy.
- The effort required to develop the special purpose languages is small in comparison to the benefits derived by their use.

To illustrate the language construction process, consider the special purpose command

```
MCINSTR(10);
```

which simulates the execution of 10 microcode instructions. This command is expanded by the

MCINSTR command expansion routine into a GPSS ADVANCE command and PL/I code for

- Calculating the execution time based upon an average instruction time,
- Accumulating the execution time for determining the processor utilization attributed to the microcode program,
- Optionally calling a trace routine, and
- Accumulating the time attributed to the execution of the current application program macro.

This command, if it occurred in the controller disk access method routine, would be expanded as follows:

```
DO;
XTIME = (CONTROLLER.MC.AITSF*10+50)/100;
#EXEC_TIME.#DISK_AM(MPTR->PROFILE#) =
  #EXEC_TIME.#DISK_AM(MPTR->PROFILE#) + XTIME;
IF TRACE.ON & TRACE.CONTROLLER THEN DO;
  IF (TRACE.MC.SELECT_MC_PROGRAMS &
    TRACE.SELECT_MC.PROGRAM.DISK_AM) |
    TRACE_MC_FUNCTION THEN
    CALL TRMCINS(CLOCK,MPTR->TERM_N,DISK_AM,XTIME);
    IF IPTR ¬= NULL & TRACE.SPPS.MACROS THEN
      IPTR->MC_ACCUM_EX_TIME =
        IPTR->MC_ACCUM_EX_TIME + XTIME;
END;
ADVANCE(*XTIME);
END;
```

The preprocessor further expands the GPSS ADVANCE command into more PL/I code and a SIMPL/I TAKE command, and likewise expands the SIMPL/I TAKE command into PL/I code.

3.4 Model Efficiency

The model simulates in detail interrupts from the loop, the host, the disk, and the interval timer. For a traffic load in which the controller handles 7 item lookup requests per second and byte interrupts from a 4800 bps loop, the model execution requires one unit of S/370 168 processing time to simulate one unit of simulated time. This is three times more efficient than a similar model written in GPSS V which was developed earlier. The execution efficiency of SIMPOS is expected to be even greater when the OPTIMIZE(TIME) option of the PL/I Optimizing Compiler is specified. During model development the NOOPTIMIZE option was used to minimize compile time.

For a typical microcode model component, the number of PL/I statements in the preprocessor output is about 4 times the number of lines of source code in the preprocessor input. On a S/370 168, typical preprocessor execution time for a microcode model component is between 6 and 10 seconds, and typical compile time using the NOOPTIMIZE option is between 20 and 40 seconds.

For a typical application program model, the number of PL/I statements in the preprocessor output is about 10 times the number of lines of source code in the preprocessor input. Thus, a 1000 line controller application model would be translated by the preprocessor into a 10,000 line PL/I program. For a program this size, typical preprocessor and compile times on a S/370 168 using the NOOPTIMIZE option are 30 seconds and 130 seconds respectively.

The preprocessor requires a 2 megabyte virtual machine. This large memory requirement is due to the loading of all command expansion routines, for GPSS, SIMPL/I and special purpose commands.

A simulation execution of a large point-of-sale system configuration also requires a 2 megabyte virtual machine.

3.5 Model Diagnostics and Debugging Aids

Preprocessor and model execution error messages for GPSS and SIMPL/I commands are contained in a diagnostic file. When a special purpose command expansion routine is correctly written, it should diagnose all possible errors in coding the command and the error messages should be added to the diagnostic file. This is the primary challenge in writing command expansion routines. Good diagnostics enhance the productivity of the modeler using the special purpose language.

It is also important to diagnose as many model execution errors as possible and put their corresponding error message in the diagnostic file. Errors not detected by the GPSS and SIMPL/I execution routines or by the routines generated as command expansions, result in PL/I diagnostic messages indicating the PL/I statement in error. The source statement in error can be determined from the compiler output listing. This is a disadvantage of PL/I GPSS when compared to GPSS V, where a source statement is indicated for all detected errors.

During the early development of the model, the standard GPSS trace facility was used to trace the GPSS transactions through the GPSS blocks. During model development, an elaborate trace facility was added which permits a model user to trace either microcode programs or application programs or both. All terminals may be selected, or only one particular terminal. The controller only, the terminals only, or the controller and terminals may be selected. When tracing microcode programs, particular microcode components may be selected. Also, GPSS block counts in the standard GPSS output have been very useful in debugging the model.

3.6 Model Output Statistics

During the early development of the model, the standard GPSS output was used as the model output. Subsequently, routines were incorporated directly into the various model components and into the generated output of the special purpose commands to gather much more detailed statistics than those provided by GPSS. Programs were written in PL/I to print these statistics. Additional statistical gathering routines and reports are planned.

3.7 Model Development Effort

SIMPOS was developed on an IBM S/370 168 under the VM/CMS operating system. The preprocessor, compile, and execution steps can be run interactively at a terminal or submitted to a batch machine.

The model, at its current level of development, required 2.5 man years of effort. The lines of code estimates below include simulation commands and PL/I statements, but exclude program comments. The development effort includes the following:

- Designing and writing the command expansion routines for 13 traffic commands, 12 microcode commands, and 46 application commands (4500 lines of code),
- Designing and writing 10 macros for the input parameter file (700 lines of code),
- Writing the 10 microcode and hardware model components (3500 lines of code),
- Writing programs for printing simulation output statistics (800 lines of code),
- Writing 3 sets of traffic, terminal application, and controller application model components (2500 lines of code),
- Writing a model user's guide, and
- Validating the model using hardware measurement data.

4. THE MICROCODE LANGUAGE

A microcode simulation language was developed and added to the preprocessor for use in the components of SIMPOS which simulate microcode and hardware. The MCINSTR command simulates the execution of a specified number of microcode instructions, and the MCTIME command simulates the execution of microcode for a specified amount of time. A set of commands simulate macros which request services from the control program. These include the POST and WAIT commands for task to task communication, the GETR and PUTR commands to "get" and "put" buffer resources, and CALL and EXIT commands for calling resident or transient microcode modules. The generated code for these commands consists of a PL/I-GPSS SWITCH command for transferring control to the control program model, a SIMPL/I-X TAG command denoting the return entry point, and PL/I statements for assigning parameters unique to the control program service request. For example, the POST command generates a PL/I statement which assigns to the transaction pointer variable TARGET_TASK the address of the GPSS transaction representing the task being posted.

The following is a simplified example of the SIMPOS program which simulates the timer access method microcode. Note the mixture of PL/I statements SELECT, WHEN, OTHERWISE, with special purpose commands WAIT, POST, and the SIMPL/I INSERT command.

```
TIMER_WAIT:
  WAIT;
  COMMAND = SOURCE_TASK->POST_COMMAND;
  SELECT(COMMAND);
    WHEN (1)   /* TIMEOUT REQUEST */
      DO;
        MCINSTR(21);      /* EXECUTE 21 INSTR. */
        INSERT(SOURCE_TASK) FIRST IN TIMEOUT_LIST;
      END;
    WHEN (2)   /* GET DATE */
      DO;
        MCINSTR(47);      /* EXECUTE 47 INSTR. */
        POST RETURN;
      END;
    WHEN (3)   /* GET TIME - EBCDIC */
      DO;
        MCINSTR(60);      /* EXECUTE 60 INSTR. */
        POST RETURN;
      END;
    WHEN (4)   /* GET TIME - BINARY */
      DO;
        MCINSTR(116);     /* EXECUTE 116 INSTR. */
        POST RETURN;
      END;
    OTHERWISE;
  END;
GO TO TIMER_WAIT;
```

5. THE TRAFFIC LANGUAGE

The following example illustrates the use of the traffic language to simulate the point-of-sale traffic generated by a particular store environment. Consider a typical mass merchandiser store which has one point-of-sale terminal located at each of several "area stations" in the store and a group of point-of-sale terminals located at the front of the store in a "checkout" mode of operation. Area station transactions have the following profile parameters:

- The average number of items per transaction is 2.
- 60% of the transactions have cash tendered.
- 40% of the transactions have check tendered.
- The clerk enters a price for 90% of the items.
- The clerk enters an item code for a price lookup request for 10% of the items.

Front-end checkout transactions have the following profile parameters:

- The average number of items per transaction is 10.
- 100% of the transactions have cash tendered.
- The clerk enters a price for 65% of the items.
- The clerk enters an item code for a price lookup request for 35% of the items.

Both profiles have the following additional parameters:

- A department number uniformly distributed between 1 and 12 is entered whenever price is entered.
- For item code entered items, 10% are coupon items, 5% are deposit items, and 5% are deposit return items, all of which are mutually exclusive.
- An item movement flag is set for all item code entered items except for deposit return items.

Also, both profiles have the following probability distributions:

- The number of items per transaction has a negative binomial distribution with a coefficient of variation of 0.75.

- Price entered items have interarrival times from a probability distribution which has a minimum of 1.5 seconds, an expected value of 4 seconds, and a coefficient of variation of .5. The distribution is a modified gamma distribution obtained by adding the minimum value, 1.5 seconds, to a gamma random variate with an expected value of (4-1.5)=2.5 seconds and a shape parameter of 1/(.5*.5)=4.

- Item code entered items have interarrival times from a modified gamma distribution which has a minimum of 1.5 seconds, an expected value of 2.5 seconds, and a coefficient of variation of .5.

- Cash tender messages have interarrival times from a modified gamma distribution which has a minimum of 5 seconds, an expected value of 20 seconds, and a coefficient of variation of 1.

- Check tender messages have interarrival times from a modified gamma distribution which has a minimum of 20 seconds, an expected value of 40 seconds, and a coefficient of variation of 1.

This example is coded below with the traffic commands. The commands are almost self-explanatory. Note that a PROFILE command references DIALOGUE commands and an ENTER command references a MSG command. The MSGVAR command establishes the name and attribute of a variable which can be assigned a value in the MSG/MSGEND command group. The SNDLEN and RCVLEN keyword parameters of the MSG command specify the message send and receive lengths in bytes, respectively.

```
TRAFFIC;

MSGVAR(DEPT,NUM);
MSGVAR(COUPON,SW);
MSGVAR(DEPOSIT,SW);
MSGVAR(DEPOSRTN,SW);
MSGVAR(ITEMMOV,SW);

PROFILE(CKOUT) DIALOGUES(CASH_CKOUT,100%);
PROFILE(AREA)
      DIALOGUES(CASH_AREA,60%,CHECK_AREA,40%);

DIALOGUE(CASH_CKOUT);
  MSGPROUP NUMBER(NEGBIN,10,.75);
    ENTER MSG(PRICE,65%) IAT(GAMMA,1.5,4,.5);
    MSGRP MSG(CODE,35%) IAT(GAMMA,1.5,2.5,.5);
  GRPEND;
  ENTER(CASH_TDR) IAT(GAMMA,5,20,1);
DIALEND;

DIALOGUE(CASH_AREA);
  MSGPROUP NUMBER(NEGBIN,2,.75);
    ENTER MSG(PRICE,90%) IAT(GAMMA,1.5,4,.5);
    MSGRP MSG(CODE,10%) IAT(GAMMA,1.5,2.5,.5);
  GRPEND;
  ENTER(CASH_TDR) IAT(GAMMA,5,20,1);
DIALEND;

DIALOGUE(CHECK_AREA);
  MSGPROUP NUMBER(NEGBIN,2,.75);
    ENTER MSG(PRICE,90%) IAT(GAMMA,1.5,4,.5);
    ENTER MSG(CODE,10%) IAT(GAMMA,1.5,2.5,.5);
  GRPEND;
  ENTER(CHECK_TDR) IAT(GAMMA,20,40,1);
DIALEND;

MSG(PRICE) SNDLEN(15);
  ASSIGN MSGVAR(DEPT) VALUE(UNIFORMI,1,12);
MSGEND;

MSG(CODE) SNDLEN(12) RCVLEN(26);
  SETON MSGVAR(COUPON,10%,DEPOSIT,5%,DEPOSRTN,5%);
  SETON MSGVAR(ITEMMOV) IF(DEPOSRTN=OFF);
MSGEND;

MSG(CASH_TDR) SNDLEN(10);
MSGEND;

MSG(CHECK_TDR) SNDLEN(12);
MSGEND;

TRAFEND;
```

6. THE APPLICATION LANGUAGE

The IBM application programming language, called "SPPS", has an instruction set similar to S/370 Assembler Language. It also contains a set of macros for common point-of-sale functions. The application simulation language was designed to correspond closely to SPPS, because application programs are frequently modified and it is much easier to change the model if the modeling language emulates the system being modeled. The EXEC command was designed to simulate the time required to execute the SPPS instructions. The command

```
EXEC SPPS(10);
```

simulates the execution of ten SPPS instructions with an average execution time specified in the input parameter file. The more explicit command

```
EXEC SPPS(MVI,(2)MVC(3));
```

simulates the execution of an MVI instruction and two MVC instructions both with a length of three bytes.

SPPS contains the structured programming macros IF, ELSE, and EIF, and also macros for reading and writing to the disk (READ and WRITE), enqueueing and dequeueing on system resources (ENQ and DEQ), calling a subroutine (CALL), scanning a table (SCAN), and many more. To simulate these macros, corresponding simulation commands were written and added to the preprocessor.

An example of a program to simulate a controller application is presented below to illustrate use of the application simulation commands. This example is a department totals update routine, which has the following logic:

- If the message is not an item sale or if there are no department totals memory buffers, then a routine is called for directly updating the department total on file. Otherwise, the department totals memory buffers are scanned for a matching department. If the department is not found, a routine is called to directly update the department total on file. If the department is found, then the department total is updated in the memory buffer.

The corresponding application simulation program for this routine follows:

```
DCLR(ITEMSALE,SW) MSGVAR;
DCLR(DEPT_TOT_BUF,SW) INPUTPARM;
DCLR(N_DEPT_MEM,NUM) INPUTPARM;
DCLR(PC_DEPT_TOT_MEM,NUM) INPUTPARM;
       •
       •
/* PUT TOTAL TO FILE IF NOT ITEM SALE OR NO */
/* DEPARTMENT MEMORY BUFFER */
IF(IF0320,ITEMSALE,OFF) OR(DEPT_TOT_BUF,OFF) THEN;
  EXEC SPPS(LA);
  ACALL CCDDUDSK PARMS(4);  /* CALL DISK UPDATE */
ELSE(IF0320);
  EXEC SPPS(MVC(2),MVC(3));
  ENQ ID(4);   /* ENQUEUE DEPT TOT BUFFERS */
/* SCAN TABLE OF BUFFERS FOR GIVEN DEPT */
  SCAN TABLEN(N_DEPT_MEM) KEYLEN(5)
    RECLEN(8) HITPCT(PC_DEPT_TOT_MEM);
  IF(IF0340,CONDCODE,NE,0) THEN;
/* IF NO BUFFER PUT TOTAL DIRECTLY TO FILE */
    DEQ ID(4);  /* DEQUEUE DEPT TOT BUFFERS */
    ACALL CCDDUDSK PARMS(4); /* CALL DISK UPDATE */
  ELSE(IF0340);
/* UPDATE TOTAL IN MEMORY BUFFER */
    EXEC SPPS((2)AB(3));
    DEQ ID(4);  /* DEQUEUE DEPT TOT BUFFERS */
  EIF(IF340);
EIF(IF0320);
```

In this example, all of the statements are special purpose simulation commands that have been added to the preprocessor. The IF, ELSE, and EIF commands simulate the application language structured programming macros. They generate PL/I IF-THEN-ELSE statements for simulating program logic and generate EXEC commands which account for the execution of TM, CLC, CLI, and BC instructions. The ACALL, SCAN, ENQ, and DEQ commands generate PL/I-GPSS SWITCH and SIMPL/I-X TAG commands for transferring control to and from the interpreter model which controls their execution.

7. CONCLUSIONS

The new IBM discrete event simulation package offers the model developer a wide range of simulation language features. This flexibility derives from the imbedding of two general purpose simulation languages, GPSS and SIMPL/I, in a full programming language. The package also contains a PL/I preprocessor which permits the development of special purpose simulation languages. The use of special purpose languages reduces model source code, provides for good model documentation, and makes a model easier to develop, maintain, and use.

ACKNOWLEDGEMENTS

I would like to acknowledge the assistance provided by my colleagues at IBM. The idea of developing the special purpose simulation languages was first proposed by Mr. M. R. Dyer of IBM United Kingdom, who implemented traffic and application languages and wrote a preprocessor for a GPSS V point-of-sale model using the S/370 Assembler Macro Language. Mr. W. J. Probeck, Dr. N. S. Strole, and Mr. J. L. Thrall developed various parts of SIMPOS. Dr. J. S. Whitlock provided very useful ideas on the organization of PL/I data structures in the model. Mr. J. Rubin has been quite helpful in answering questions concerning PL/I GPSS, SIMPL/I X, and the PL/I preprocessor.

References

Bryant, R. M. (1980),"Discrete Event Simulation Languages", Proceedings of 1980 Winter Simulation Conference, Orlando, Fla., Dec. 1980, pp. 25-40.

Fishman, G. (1978), Principles of Discrete Event Simulation, John Wiley and Sons, New York.

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

IBM Corporation (1977), General Purpose Simulation System V User's Manual, Publication SH20-0851-2, Data Processing Division, White Plains, New York.

IBM Corporation (1979), PL/I Language Construction Preprocessor Program Description/Operations Manual (Program Number: 5796-PLL), Publication SH20-2164-0, Data Processing Division, White Plains, New York.

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.

Rubin, J. (1981),"Imbedding GPSS in a General Purpose Programming Language", Proceedings of 1981 Winter Simulation Conference, Atlanta, Ga., Dec 1981.

Shannon, R. E. (1975), Systems Simulation, The Art and Science, Prentice Hall Publishing Company, Englewood Cliffs, New Jersey.

Shub, C. M. (1980),"Discrete Event Simulation Languages", Proceedings of 1980 Winter Simulation Conference Volume 2, Orlando, Fla., Dec. 1980, pp. 107-124.