CLASS

COMPOSITE LANGUAGE APPROACH FOR SYSTEM SIMULATION

(A TUTORIAL)

Harold G. Hixson

Computer Sciences Division (ACTR)
Headquarters, Air Force Logistics Command
Wright-Patterson Air Force Base, Ohio 45433

Summary

There are several possible approaches to providing both discrete event and continuous system simulation capability in the same modeling context. One approach is to develop a new language which has this capability. Another, which is addressed in this tutorial, is the use of compatible languages in combination. One possibility[4] involves the use of SIMSCRIPT II as the basic language. Another possibility is the use of the General Purpose Simulation System (GPSS) and its "HELP" language FORTRAN. This tutorial explains the basic capabilities required for discrete event modeling, those needed for continuous system simulation, and the considerations involved in achieving a "hybrid" language capability. It also includes application examples, concentrating on an application requiring additional language facilities for modeling process control computer systems.

Introduction

Background

A recent paper[4] described the union of the SIMSCRIPT II language and GPSS concepts to provide capability for both discrete event and continuous system simulation. It is possible to use the combination of GPSS and FORTRAN for the same purpose and that possibility is addressed here. A "natural" application for this simulation capability is the modeling of process control computers. However, general purpose simulation languages need augmentation so that they are adequate for modeling computers. For example, SIMSCRIPT II is the host language for ECSS[7] (Extendable Computer System Simulator). Similarly, a recent development effort utilized GPSS as the host language for implementing concepts inherent within the System and Software Simulator (S3)[6]. These two examples illustrate the possibility for a vertical relationship between languages (building on a basic language in "macro" fashion) as opposed to a horizontal relationship between languages (permitting communication and transfer of control between programs written in two or more languages).

Tutorial Organization

The first section of this tutorial is designed to impart a basic understanding of discrete event simulation in two of its most popular forms - GPSS and SIMSCRIPT. The second section provides descriptions of the basic logical, computational, time dependent, and special elements necessary for continuous system simulation. The third section presents the major considerations for achieving full communication between, and compatibility of, the sub-models in a combined discrete event and continuous system simulation. The fourth section describes several very simple examples of CLASS applications. The fifth section concentrates on one of the most challenging applications of CLASS - a process control computer system. The last section addresses language needs, GPSS adaptations, and preprocessor capabilities which are required to make this application practical.

Basic Simulation Concepts

Discrete Event Simulation

Figure 1 summarizes the basic capabilities possessed by GPSS and SIMSCRIPT. In general, the conditions under which the use of one is preferred over the use of the other are specified in Figure 2. Discrete event simulation features event occurrences at irregular intervals.

Continuous System Simulation

Figure 3 summarizes the basic capabilities possessed by most continuous system simulators. These operational elements appear in categories defined by functional similarities. Continuous system simulation features regular time incrementation with periodic state change throughout the model. This regularity is indicated by the term "sampled-time simulation".

The Ties That Bind

A continuous sampled-time simulation can be exactly represented as a discrete event simulation by defining a special event which computes the outputs of each simulated "analog" block in the continuous portion of the model at every sampled time instant. This technique is used in the CLASS approach and allows the discrete event and continuous sections of the model to interact on a global basis. One of these possible interactions is that of allowing output variables from the discrete portion of the model to become input variables to the continuous subsystem (and vice versa). Modification of continuous block parameters by a discrete event process allows re-specification of a block's characteristics

during the course of a simulation. The use of continuous block forced state change allows a discrete event to cause a step change in an otherwise continuously changing variable. The process of entity transformation models material flow between continuous and discrete event storage areas. When the value of a specified continuous variable passes a predetermined threshold a discrete event may be triggered, either immediately or after a time delay. Continuous sub-model structure change may allow substitution, addition, or deletion of blocks with appropriate adjustments in their interconnections. These adjustments may be activated by a discrete event and thereby allow system self-organization or special effects.

Process Controller Applications of CLASS

Application Description

A variation of the CLASS approach has been used to simulate a process control computer (the IBM 1800), which can be used to control and measure the performance of a number of aircraft engines in various stages of trial (ground) operation in test cells. The details of this application were provided in a recent paper[4] and hence are not repeated here. In addition to the original implementation described in that paper, this application is under study using a combination of GPSS and FORTRAN.

Continuous System Simulation Submodel

The aircraft engines in this application are being simulated with a Continuous System Simulation Language (CSSL) implemented in FORTRAN. Communication with the FORTRAN subroutine required is through the GPSS HELP block and its arguments (Standard Numerical Attributes). The integration technique used in the submodel is the second order predictor-corrector[8]. A sorting method[9] is utilized to initially place the elements in the correct sequential order to permit solution. Interpretive execution of the logic and computation specified by these elements produces output which is arranged in time series tabular form. This output is printed throughout the duration of the CLASS simulator execution.

Discrete Event Submodel

The logic of programs used by the process control computer are described by GPSS blocks. The components of the computer are represented by either GPSS facilities or storages, as appropriate. However, for obvious reasons, the capability of GPSS needed to be enhanced through special adaptations for this application. This enhancement is addressed in the following section.

Language Adaptations and Supplements

Computer System Modeling Needs

The need for a powerful, efficient, flexible, responsive, and open-ended simulation language for studying computer systems has been universally recognized. However, completely documented and tested languages which approach this ideal have been slow to appear. Two languages show great promise but the success of each depends on further development, extension, and refinement. The System and Software Simulator (S^3) had neither a rich syntax nor reasonable limitations on the size of computer configurations to be simulated in its first full implementation on the UNIVAC 1108 computer. The recently developed Air Force version of S^3 for the Honeywell (GE) 615 and the commercially available (proprietary) System Analysis Machine (SAM) have both overcome these limitations to some degree. The second language which has been developed to fill these needs is the Extendable Computer System Simulation (ECSS). The commercially available simulation algorithm packages used widely for computer configuring and costing are not considered here because each of these is pre-programmed and parameter-driven as opposed to being a language for compilation or pseudo (interpretive) language. It should be noted that the composite language described here was developed and used prior to the availability of the 615 version of S^3, SAM, or ECSS. As such it was intended solely for use as an interim technique. However, its success in simulating large scale multiprocessors and small scale process controllers indicates that it employs a sound approach and that therefore a more efficient implementation of this approach may be worthwhile. In such an implementation, the syntax could be modified to reflect "natural" computer terminology.

GPLUS

GPLUS is a completely obscure acronym for the composite of the "GPSS Language Using S^3 Concepts" and serves only to provide a concise name for this approach. Common functions of computer operating systems[5] and application programs are listed in Figure 4 along with a few of the GPSS blocks used in simulating these functions. A sample of the similarities between most of these blocks and corresponding original S^3 statements is provided in Figure 5.

One of the primary mechanisms of GPLUS (which enables it to take on the form of S^3) is that of recursion. Contiguous parameters of each GPSS transaction are utilized in creating a pushdown stack of GPSS subroutine return block numbers. Ordinarily, two parameters are required for each level of subroutine call, one for the identification number of the subroutine, and the other for the location where the transaction departed that subroutine. The transaction will eventually return to the block immediately following that location.

Figure 6 describes the GPLUS logic for simulating individually activated load modules and subroutines. Simulated programs may be reentrant, conditionally reusable, serially reusable or non-reentrant. GPLUS provides for loading a reentrant module only prior to its first use and for loading a new copy of each non-reentrant module for every activation. It also provides for special loading conventions and user queuing for reusable modules. GPSS subroutines are used to describe the logic of program modules which execute synchronously. The GPSS "SPLIT" block creates transactions which are sent to GPSS subroutines which represent load modules which execute

asnychronously. Special events which occur only when the last one of a family of asynchronously executing load modules terminates are triggered by GPSS blocks which route the related GPSS transactions to one common ASSEMBLE block.

A few of the possible output statistics from GPLUS are listed in Figure 8. These are used to analyze processor, channel, and device utilization and identify under-utilization, imbalanced utilization between similar components, or saturation. Program execution statistics indicate areas within software modules or application programs where revised logic might result in more efficient (faster) execution. Analysis of job or activity queuing may result in useful hypotheses concerning workload management or proper settings for software control variables. A system-wide review of the statistics may indicate revisions to the system configuration, such as cross-barring channels, standardizing memory partition size, etc. If adjustment does not provide sufficient performance improvement, system augmentation may be necessary. After deciding what actions may be desirable, the altered system may be simulated to test and validate its predicted performance improvement.

The Need for Readily Available Models of Computer Systems. When computer component capacity and rated performance factors, configuration connections, software descriptions, and application program logic must all be collected from diverse sources for a large computer system, preparations for a simulation study can be excessively time consuming. When the time necessary for manually coding the simulation language statements is added, the resulting manpower resource requirement may either become prohibitive or impose an acceptable time delay before obtaining simulation results. As an aid in minimizing the time and manual effort required for obtaining one of the items listed above (the simulation logic for application programs), the following technique was developed.

SIGNAL. The S^3 Input Generator for All Languages (SIGNAL) is a decompiler/recompiler. It is used in the process of transforming source language programs into either original S^3 statement form or the GPSS blocks needed to implement GPLUS (at the option of the user). It currently allows the user to choose either FORTRAN IV or COBOL as input. The current implementation is in FORTRAN for the Honeywell 615 computer and depends on the utilization of the compilers for that computer. The first step of the process is to compile the source language. The object code produced by the compiler, a list of computer (binary) operation codes and associated timings, and a list of input/output operations and their associated average times are input to SIGNAL. SIGNAL decompiles the object code, computes timings of instruction execution, and translates looping and I/O operation codes to their equivalent GPSS blocks. Instruction and code words are counted and incorporated into the simulated program description. This simulated application program description, in terms of GPSS blocks, is punched out in card form. The user selects one of a number of large-scale computers to which this simulation input applies by means of a control variable which is simply the model number (e.g., 615, 635, 6000, 1108, 1110, 6600, 7600, 155, 165, etc.). In addition, upon user request the total program execution time and an estimate of program I/O time for each of these computers is printed. The I/O time estimate assumes average seek time, average latency time, no I/O request queuing, and no channel, device, or read/write head contention. Figure 7 is an example of application program statement input to the compiler and the corresponding simulation program statement output from SIGNAL.

BIBLIOGRAPHY

1. D. Fahrland, "Combined Discrete Event/Continuous Systems Simulation," SRC-68-16, Systems Research Center, Case Western Reserve University, July 1968.

2. D. Fahrland, "Combined Discrete Event/Continuous Systems Simulation," Simulation, Volume 14, Number 2, February 1970.

3. H. Hixson, "Equipment Maintenance Studies Using a Combination of Discrete Event and Continuous System Simulation," Proceedings of the Third Conference on Applications of Simulation, December 1969.

4. H. Hixson, "CLASS - Composite Language Approach for System Simulation," Proceedings of the Fourth Conference on Applications of Simulation, December 1970.

5. Leo J. Cohen, "Operating System Analysis and Design," Spartan Book, 1970.

6. Leo J. Cohen, "System and Software Simulator," Documents AD679269 through AD679272, Clearinghouse, U.S. Department of Commerce, 1968.

7. N. R. Nielson, "ECSS, An Extendable Computer System Simulator," RM-6132-NASA, The RAND Corporation, February, 1970.

8. D. D. McCracken and W. S. Dorn, "Numerical Methods and FORTRAN Programming," John Wiley and Sons, New York, 1964.

9. M. L. Stein and J. Rose, "Changing From Analog to Digital Programming by Digital Techniques," Journal of the Association for Computing Machinery, 7, 10-23, 1960.

BIOGRAPHY

Harold G. Hixson is the Group Leader of the Numerical Methods and Information Systems Group in the Headquarters of the Air Force Logistics Command at Wright-Patterson Air Force Base, Ohio. He has held positions at that location as an actuary, mathematician, and operations research analyst. Mr. Hixson graduated with a B.S. in Mathematics from Otterbein College in 1957. His professional affiliations include Simulation Councils, Inc. and the Association for Computing Machinery.

GPSS	SIMSCRIPT
TRANSACTIONS/PARAMETERS	TEMPORARY ENTITIES/ATTRIBUTES PERMANENT ENTITIES/ATTRIBUTES
ASSEMBLY SETS GROUPS OF TRANSACTIONS CHAINS OF TRANSACTIONS (SYSTEM/USER)	SETS OF ENTITIES
BLOCKS	SETS OF EVENT NOTICES EVENTS EVENT NOTICES EVENT ATTRIBUTES EVENT ROUTINES
HELP SUBROUTINE MACROS	CALLED ROUTINES INVOKED FUNCTIONS
QUEUES FACILITIES STORAGES	
SYSTEM NUMERICAL ATTRIBUTES	TIME/RANDOM NUMBERS
SAVEVALUES	SYSTEM VARIABLES LOCAL VARIABLES
FUNCTIONS	RANDOM LOOK-UP TABLES
VARIABLE STATEMENTS	STATEMENTS
SWITCHES	

Figure 1

Comparison of Basic Language Features

USE GPSS FOR:	USE SIMSCRIPT FOR:
QUEUING PROBLEMS	COMPLEX NUMERIC, EVENT, AND/OR SET-ORIENTED PROBLEMS
SMALL TO MEDIUM APPLICATIONS	LARGE APPLICATIONS REQUIRING EFFICIENT MEMORY USE
LITTLE INPUT AND STANDARD OUTPUT SUFFICIENT	SUBSTANTIAL INPUT AND/OR TAILORED OUTPUT REQUIRED
NUMERIC PRECISION AND MAGNITUDE REQUIREMENTS NOT SEVERE	PRECISE AND/OR LARGE NUMBERS REQUIRED
PROGRAMMING EFFORT AT NOVICE LEVEL, SHORT DURATION AND/OR INDIVIDUAL APPROACH	PROGRAMMING EFFORT AT EXPERT LEVEL, LONG DURATION, AND/OR TEAM APPROACH
FAST COMPUTER EXECUTION NOT REQUIRED	FAST COMPUTER EXECUTION REQUIRED
FREQUENT MODEL MODIFICATION EXPECTED	INFREQUENT MODEL MODIFICATION

Figure 2

Comparison of Language Applicability

Basic Single Values:

- INDEPENDENT VARIABLE
- CONSTANT
- ABSOLUTE VALUE

Arithmetic Operations:

- SUMMER (+)
- MULTIPLIER
- DIVIDER

Roots and Powers:

- SQUARE ROOT
- LOGARITHM
- EXPONENTIAL
- POWER OF VARIABLE

Trigonometric Functions:

- SINE
- COSINE
- TANGENT
- ARC SINE
- ARC COSINE
- ARC TANGENT

Calculus Operations:

- INTEGRATOR
- DERIVATIVE

Logical Elements:

- AND/NAND/NOT
- EOR/IOR/NOR
- EQUIVALENCE
- RESET (Flip-Flop)
- COMPARATOR

Switches (Relays):

- INPUT SWITCH
- OUTPUT SWITCH
- FUNCTION SWITCH
- BANG-BANG
- DEAD SPACE
- LIMITER
- NEGATIVE CLIPPER
- POSITIVE CLIPPER

Time Dependent Operations:

- DELAY (lag)
- IMPULSE
- PULSE
- RAMP
- STEP
- HYSTERESIS

Special Operations:

- STORE/ZERO-ORDER HOLD
- MAXIMUM/MINIMUM
- QUANTIZER
- FUNCTION GENERATOR

Figure 3

Continuous System Simulation Features

Function(s)	Location	Type(s) of GPSS Block(s)
Transaction Input	AP	GENERATE
Receiver, Director, Allocator, & Loader	OS	Many Blocks
Instruction Timing	AP and OS	ADVANCE
Transfer Control	AP and OS	TRANSFER
Looping	AP and OS	LOOP
I/O Instructions	AP and OS	TRANSFER (SBR)
Activate Program Module	AP and OS	TRANSFER (SBR)
Call or Perform Subroutine	AP	TRANSFER (SBR)
Multiprogramming Dispatcher	AP relinquishes, OS controls and dispatches next AP	TRANSFER (SBR)
I/O Request Processing	OS	Many Blocks
Distributor	OS	Many Blocks
Deallocator and Terminator	OS	Many Blocks

Figure 4

Application Program (AP) and Operating System (OS) Functions

Partial List of Original S^3 Instructions		Equivalent GPSS Blocks	
TRA	LABEL	TRANSFER	,LABEL
TRA-P	PERCENT, LABEL 2	TRANSFER	FRACTION, LABEL 1, LABEL 2
READ (OR WRITE)		TRANSFER	SBR,(READ / WRITE),1
MOVE	CHARACTERS	ADVANCE	TIME
MATH	OPERATIONS	ADVANCE	TIME
COMPUT	INSTRUCTIONS	ADVANCE	TIME
ETC.			
		LABEL ASSIGN	PARAMETER, ITERATIONS
LOOP	ITERATIONS,LABEL		
		LOOP	PARAMETER, LABEL
CALL	SLM,DLY	TRANSFER	SBR,SLM,1
CALL	ALM,NODLY	SPLIT	1,ALM
PLACE*	AT,Q	LINK	Q,FIFO
SELECT**	AT,Q	UNLINK	Q,LABEL,1

* This instruction places the available transaction (AT) in a queue called Q.

** This instruction selects a transaction (AT) from a queue called Q.

Figure 5

Equivalence of Some S^3 Instructions and GPSS Blocks

Type of Application Program Simulated	Type of Execution Simulated	Calling Program Contains This GPSS Block	Called Program Ends With This GPSS Block
Load Module (e.g. ALM)	Asynchronous	SPLIT 1,ALM	TRANSFER ,TERM
		Note: This ending assures that the the copy transaction is destroyed after module execution, resource deallocation, and program termination.	
Load Module (e.g. SLM)	Synchronous	TRANSFER SBR,SLM,1	TRANSFER P,1,1
		Note: Parameter 1 contains the TRANSFER Block number.	Note: This routes the transaction back to block following TRANSFER SBR,SLM,1
Subroutine (e.g. SUB)	Sequential	TRANSFER SBR,SUB,1	Series of GPSS blocks to implement recursion by return address list and pointer.

Figure 6

Simulation of Forked Activity Programs and Subroutines

COBOL Source Statements (Input to the Compiler which Produces Input for SIGNAL)	GPLUS Simulation Blocks (Output from SIGNAL)	
MOVE MAXDAYS TO PERIOD.		
MOVE 99 TO TYPE.	ADVANCE	283
WRITE EXT01.	TRANSFER	SBR,WRITE,1
	ADVANCE	161
IF X GREATER THAN 6 GO TO TWO.	TRANSFER	.5,,TWO
DIVIDE 1000 INTO SEC GIVING MILLI.		
MOVE TIME TO LOG.	ADVANCE	341
TWO.		
ADD 1 TO YEAR GIVING NEXT YEAR. TWO	ADVANCE	107
READ INPUT	TRANSFER	SBR,READ,1

Figure 7

An Example of Source Language Translated to GPSS Blocks

---

* Processor, Memory, Channel, and Device Utilization
* Processor and Elapsed Time by Program
* Number of Real Time Transactions Processed
* Number of Program Executions
* Distribution of Activity Within Each Program
* Number of Program Swaps
* Job and Job Segment Throughput
* Job and Job Segment Queue Lengths and Waiting Time Distributions for Each Operating System Queue
* I/O Request Queue Lengths and Waiting Time Distributions

Figure 8

Types of GPLUS Output Statistics