A MODEL OF SAFEWAY'S CONTROL SYSTEM

John Ryder
Safeway Stores, Incorporated
Data Processing Department
Oakland, California

Abstract

A detailed model, using GPSS/360, of the extensive software designed and implemented to support Safeway's Basic Management Information System. Also modeled, to the same level, are most of the functions of OS/MVT. HELP routines are used to implement external (console) control and display capabilities, dynamic allocation of GPSS matrices, paging of GPSS blocks and common, and other functions designed to reduce the amount of permanently resident data.

1. INTRODUCTION

This paper discusses development of a simulation model for Safeway's Basic Management Information System (BMIS). Safeway's BMIS undertakes to support 18 retail divisions across the United States with centralized remote-batch data processing. At this writing, one division is partially supported and further conversion is underway.

All software is written and all data files are maintained at central. Programs to read data cards or paper tape and print reports are transmitted from central to the terminals, based on the format of the data to be transmitted. Processing requests are initiated at the terminal via a single run description card followed by the input data. The central control system generates a data set consisting of the input data, after which a job to process it is either immediately initiated or queued for future initiation. Job initiation consists of selecting and completing disk-stored skeletonized JCL and passing it directly from core to a reader/interpreter. Application programs then process the data and generate reports ("outbound" data sets) which are queued to be transmitted by the control system to the specified terminal (or terminals) as soon as possible.

The current central hardware configuration consists of an IBM 360/50 with 512K fast core and one megabyte LCS; one selector channel supporting a 2314 plus a 2803 with seven 2401 model 2 tape drives (six 9-track, one 7-track); multiplexor channel supporting a 2540 card reader/punch, a 1403-N1 printer, a 2701 communications adapter and a 1052 console typewriter. The entire system with the exception of the tape drives, is duplexed, with complete component switching capability. The terminal is a 360/20 with a 1403 model 2 printer and a 2501 card reader, connected

to central by leased telephone lines and 4800 bps Milgo modems.

Software, as described above, operates under OS/360 MVT, with some local modifications. The term "Safeway Control System" (SCS) is construed to include locally written control system software and OS/360, and to exclude application programs.

The overall project began in 1966. The simulation project, consisting mainly of the author, did not start until June 1968, after initial system design was complete and the control system written and operational. What has been modeled, therefore, is largely an existing system, providing obvious advantages in the areas of design and validation.

The model is only now approaching the end of its initial development stage and the beginning of its use as a management decision making tool. In light of this, the paper will deal largely with the scope of the model, its intended uses, and particular techniques devised in the course of its development. Practical results remain in its future. The reader is assumed to have a reasonable knowledge of both GPSS and IBM's Operating System/360.

2. OBJECTIVES

2.1 SHORT TERM

It is known that the current configuration, which adequately supports a single division, will certainly not support very many. Simulation is expected to predict the overload points of the various components in the system, and at such points attempt to find solutions within the configuration; e.g., altering the distribution of data sets on the direct access devices to reduce channel contention. If such a solution cannot be found, the next step is to evaluate alternate configurations until the system is once again supportable. Alternately, software bottlenecks should be detectable, from which design changes can intelligently be suggested.

2.2 LONG TERM

The system, while in operation, is far from complete. Many applications remain to be written; others are known to need major revision. The control system itself is being constantly upgraded to meet new demands, and will be continually modified to take advantage of knowledge gained from operational experience and to incorporate new features to support future demands (indeed, the entire BMIS project is intended as a first step toward a more comprehensive management information system).

It should be possible to incorporate new design features into the model and evaluate their effect on system performance. Similarly, different hardware components should be configurable as additions to or replacements for existing components.

3. RESOURCES

The model is written in GPSS/360, chosen for its flexibility, generality, and availability. The author had done some previous work with it, and thus familiarity was an additional factor. CSS/360 was investigated since a computer system was to be modeled, and many of the built in features appeared desirable. Unfortunately, the restrictions implicit in the division of supervisor- and application-modeling within the language made the task of modeling a control system that lay somewhere between the two impractical if not impossible.

Development of the model to its present state required approximately 18 man-months during one calendar year, personnel as follows: staff programmer, 1 man year (author); senior programmer, 3 man-months; programmer, 3 man-months (these personnel designations, probably meaningless to the reader, represent respectively

7 years, 3-1/2 years and 1-1/2 years of programming experience). The model contains approximately 4100 blocks, occupies a region of 330K, required some 500 hours of computer time to develop.

Running time of the model (excluding assembly time) varies drastically depending on the load placed on it: an elapsed time-to-simulated time ratio of 10 to one can be expected with a single terminal being modeled; with eight or ten terminals modeled the ratio is as high as 30 or 40 to one.

5. REPRESENTATION

OS/MVT tasks are modeled as transactions which compete for facilities representing such system resources as I/O channels and devices, communication lines, and the CPU. A task may be represented by more than one transaction simultaneously, as when the overlapping of instruction execution with I/O is being modeled. A clock unit of one millisecond drives the model, chosen as the largest increment still permitting timings at the program logic level to be modeled (used in conjunction with a scheme of accumulating fractions of a millisecond for each task).

Transactions are directed onto user chains in certain queueing situations (such as I/O channel contention), and under conditions where they would otherwise have to remain on the current events chain in an interrupt status for extended periods of time (e.g. a low-frequency task entering a voluntary wait state.)

Most OS/MVT control blocks are modeled as halfword matrices, facilitating a detailed parallel to the real system of the information and frequently complex chaining associated with such control blocks (e.g., TCB's, DCB's, RB's, IOB's, ECB's). Matrices also model such things as data sets (their location and extent), I/O device status (volume mounted, head position for direct access) and core storage allocation. Logic switches represent resource status for a variety of enqueue/dequeue functions.

6. TECHNIQUES

6.1 INITIALIZATION

When a simulation run begins, control passes immediately to a HELP routine that provides the initialization phase, reading and processing several input data sets.

(1) Cards describing the data sets to be permanently allocated (data set identification, volume serial number, cylinder at which data set begins, and how many cylinders it occupies). The information is validated and entered into the matrix describing data sets.

(2) Cards specifying which volumes are to be mounted on which devices are used to initialize a matrix used by IOS when servicing I/O requests.

(3) Cards requesting initialization of matrices representing the areas of direct access devices that are unused (used and updated by SCRATCH and ALLOCATE routines; determines placement of data sets created during the simulation).

(4) Cards describing the configuration of communication lines and terminals, which will cause activation of transactions for the specified terminals and the ATTACHing of appropriate control system tasks to handle them.

(5) Cards describing the "runs" to be submitted from specific terminals at times specified on the cards. From there a control data set is built from which "runs" will be selected at the appropriate times during the simulation and "submitted" from the specified termi-

nals. The cards provide both ease of variation and ease of repetition of conditions.

6.2 DISPLAY AND CONTROL

A subtask is ATTACHED which operates independently, responding to console commands, permitting external observation and, if desired, control of the simulation run.

6.2.1 Control

The subtask selects the "runs" to be submitted and makes the data available to the terminal transaction. In order to set up the "runs" at the correct times, the subtask CREAT's a transaction and places it on the future events chain, with its departure time equal to the time the next "run" is to be submitted. At this time, the transaction enters a HELP block, causing the subtask to be POSTed. The following external controls can be imposed on the simulation during execution on command:

(1) the value of any half- or full-word savevalue may be altered

(2) the value of any entry in a half- or full-word matrix savevalue may be altered

(3) a "run" to be submitted from a terminal may be inserted in the queue established during initialization

(4) the model may be checkpointed at any time (normal GPSS SAVE - a transaction on the future events chain has its block departure time changed to current time plus one and is rechained on the future events chain. When the transaction is placed on the current events chain it enters a terminate block reducing the terminations-to-go count to zero), after which it continues

(5) the model may be shut down at any time, involving a checkpoint, as in (4), a print-out of the final time, time ratio, and active job displays, and a termination of the run. (Restart is simple - the same deck is used minus the input cards described above.)

6.2.2 Display

The following information can be displayed on command:

(1) internal simulator current time

(2) the ratio of elapsed time to simulated time

(3) status of model jobs in execution (similar to the OS/360 command "D A", with the addition of actual CPU time used by each job)

(4) current statistics for any facility in the model

(5) the value of any half- or full-word savevalue

(6) the value of any entry in a half- or full-word matrix savevalue

(7) counts for any block in the model

(8) counts for the block with the greatest total entry count

(9) counts for the n blocks with the greatest total entry count, $n \leq 15$

(10) counts for the n blocks with the greatest total entry count, $n \leq 15$, within a specified range of blocks (very useful in debugging when a loop is suspected)

(11) an interval may be started and later

terminated for a facility, and at the termination the utilization of the facility during the interval is printed together with the beginning and ending time of the interval. Up to six intervals may be in progress at any one time, and the same facility may be measured in overlapping intervals.

Automatically displayed by calling a HELP routine at the appropriate times are the beginning and ending times of each job step modeled. The display also includes the CPU time used by the job/job step. Commands and displays are written to an output data set that is printed at the termination of the simulation and (optionally) on the console typewriter.

6.3 JOB REPRESENTATION

Each job to be modeled is represented in a pair of data sets maintained on direct access storage. These data sets describe the characteristics of the job, e.g., the number of steps, region size of each step, number and nature of the data sets required by the job, the number of JCL cards necessary to initiate the job, job priority, etc. The data is extracted at various times by HELP routines to control the modeled job.

6.4 DYNAMIC ENTITY ALLOCATION

Since the arrival of job requests cannot be internally predicted and the same job may run simultaneously on requests from more than one terminal, the quantity and control of certain GPSS entities becomes prohibitive if an effort is made to pre-establish them. The problem is overcome by the use of HELP routines to dynamically allocate and deallocate matrices on an as-needed basis, reducing the number of matrices (as well as the common associated with them) to the maximum needed at any one time, which is far smaller than the total required. A "prototype" matrix may be optionally copied into a newly allocated matrix. Logic switches present a similar problem, solved by reserving a pool of them to be allocated in a way similar to matrices.

6.5 PAGING

Matrices and logic switches are the second and third most used entities in the model; the most used, naturally, are blocks. These, too, get out of hand when it is considered that a hundred or more jobs may be modeled. Relatively few of these can be "executing" at any one time (restricted by the amount of core and the number of initiator/terminators modeled), although any one of them must be available for selection.

This problem, as well as some others, is resolved through use of a paging technique:

All models of application programs are written and assembled external to the main model (the "main model" consisting of OS/MVT, the control system and any non-block entities needed by an application model). An assembly environment is provided to include EQU's, SYN's and MACRO's from the main model, plus a transfer table to permit use of the common functions available in the permanently resident part of the model.

When an application program model ("submodel") successfully assembles, the execution phase invokes a HELP routine that copies the blocks and their associated common into a page data set, together with a relocation dictionary to resolve the submodel's internal references (references to the main model are "absolute" and need no relocation). Each page is assigned an identification number during its assembly, and this ID becomes part of the description of the job step (or steps) that will use the submodel.

When the main model arrives at the equivalent of program fetch for a submodel, a page control matrix is scanned to determine whether the page

required is already in core. If it is, a use count for the page is incremented and the page is entered immediately. If it is not, a HELP routine is called to fetch the page, with one of several possible results:

(1) The page is found, space (blocks plus common) is available to contain it; the page is then read in, its references relocated, and an entry added to the page control matrix describing it, after which control returns to the main model, which permits a transaction to enter the submodel.

(2) The page is found, but space is not available to contain it; the routine will then purge any page in core with use counts of zero and try again to find space for the requested page. If sufficient space is now available, the logic proceeds as in (1). If space is still not available, control returns to the main model with an indication of the condition. The main model will defer the page request until such time as space becomes available.

(3) The page is not found; control returns to the main model with an indication of the condition, at which point the main model simulates an immediate exit from the submodel, and the simulation continues.

The only limiting factor on the number of pages that may be simultaneously resident is the amount of space provided in the main model for this purpose.

Subsidiary benefits of paging are:

(1) Greatly accelerated debug time - a submodel assembly should require approximately 5 minutes and a 100K region, as opposed to assembling the main model, which requires approximately 90 - 120 minutes and a 350K region.

(2) Each submodel can be developed independently, greatly reducing a potentially severe coordination problem.

(3) Minimal impact on the main model resulting from the additions of new submodels and submodel errors and changes.

6.6 I/O INTERRUPTS

I/O interrupt handling is facilitated by having all ADVANCE's, while in control of the CPU, executed in a common subroutine called by a GPSS MACRO. An I/O interrupt is modeled by a transaction leaving the future events chain at the time the interrupt is scheduled to occur. This transaction PREEMPT's the CPU on a priority basis (itself having a priority greater than any normal CPU user), removing any transaction then SEIZEing the CPU and directing it to rejoin the queue of transactions waiting for the CPU. The time remaining for it to ADVANCE is placed in the parameter referenced by the common ADVANCE block and reserved in all transactions for this purpose. The transaction will eventually be removed from the queue by the Task Dispatcher, on the basis of its priority, and directed to return to the common ADVANCE block to complete its use of CPU time.

7. STATISTICS GATHERED

(1) Overall utilization of facilities (CPU, I/O channels and devices, communication lines) and core storage.

(2) Queueing statistics for I/O channels and devices, CPU

(3) Incidence of direct access device arm contention and data set interference.

(4) Tabulation of the load placed on the CPU by I/O operations, including the occurrence of overruns.

(5) Throughput times for each job modeled plus CPU utilization of each job and each step within the job.

(6) Number of task switches, both intra-region and inter-region

(7) Utilization of various OS/MVT and control system functions.

8. VALIDITY

In the course of model development many OS/MVT functions were timed in controlled environments. Results were used directly in modeling some functions in limited detail, indirectly in verifying preliminary results of functions modeled in greater detail. Other functions, of both OS/MVT and the control system, not easily susceptible to direct timing were "hand-timed" by inspection of the source code. This form of timing is particularly accurate when functions are modeled to the program logic level. Some overall verification was provided by coding a test run to be submitted "live" from a terminal, a submodel of the test run, and comparing the results. Tests are underway with a submodel of a representative application program. Results to date, although preliminary, have shown no significant discrepancies. Future tests will include the use of AMAP to provide detailed statistics on internal system functions and run characteristics.

9. APPENDIX

9.1 OS/MVT FUNCTIONS MODELED

9.1.1 SVC's

(1) First and second level interrupt handlers

(2) Transient areas

(3) Transient area handlers, fetch and refresh routines

(4) Link-pack area

9.1.2 Initiator/Terminators

(1) Data set allocation and deallocation

(2) System resource enqueueing and dequeueing

(3) Region allocation and deallocation

(4) ATTACHing and DETACHing of job step tasks

9.1.3 Space Management

(1) Process requests for region allocation and deallocation by I/T's

(2) GETMAIN/FREEMAIN

9.1.4 Task Management

(1) Dispatcher

(2) Ready queue (priority-ordered TCB chain)

(3) ATTACH/DETACH, WAIT/POST

9.1.5 Data Management

(1) Queued and direct access methods

(2) OPEN/CLOSE, READ/WRITE/CHECK, GET/PUT/RELSE, SETL/ESETL

9.1.6 I/O Supervisor

(1) Priority queueing

(2) Channel and device enqueue/dequeue

(3) Channel program execution

(4) I/O interrupt handling

9.1.7 Direct Access Device Space Management

(1) Dynamic allocation and scratching of data sets

(2) Unit classes

9.1.8 Other OS/MVT Entities Modeled Include:

TCB's, RB's, ECB's, DCB's, IOB's, Channel Programs, I/O Buffers, UCB's

9.2 HARDWARE MODELED

9.2.1 CPU

9.2.2 Core Storage - Both "Fast" Core and LCS

9.2.3 Cycle Speeds - A Combined Characteristic of 9.2.1 and 9.2.2; Two Speeds - One for "Fast" Core, One for LCS

9.2.4 I/O Channels

9.2.5 I/O Devices

(1) 2314

(2) 2301

(3) 2401 - 2

(4) 2401 - 5

(5) Others may be added with relative ease

9.2.6 Communications Lines (To Terminals)

BIOGRAPHY

The author received his BA in Mathematics at the University of Florida, and in 1962 joined IBM. At Houston, Texas, he designed and developed real time control programs in support of NASA's Gemini and Apollo missions. In 1967 he joined Safeway as a systems programmer, where he participated in the development of Safeway's Control System. Since mid-1968 he has worked on the GPSS model of this system, which is the topic of this paper.