REPORT NO. DOT-TSC-FAA-71-23

# COMPUTER SYSTEMS PERFORMANCE MEASUREMENT TECHNIQUES

J. GERTLER, H. GLYNN,
V. HOBBS, F. WOOLFALL
TRANSPORTATION SYSTEMS CENTER
55 BROADWAY
CAMBRIDGE, MA. 02142


JUNE 1971
TECHNICAL REPORT



Prepared for
DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION
800 INDEPENDENCE AVE., S. W.
WASHINGTON, D. C. 20546

TECHNICAL REPORT STANDARD TITLE PAGE

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| DOT-TSC-FAA-71-23 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| Computer Systems Performance Measurement Techniques | June 30, 1971 |
| | 6. Performing Organization Code: TCC |

| 7. Author(s) | 8. Performing Organization Report No. |
|---|---|
| Judith Gertler, Herbert Glynn, Vivian Hobbs, Frederick Woolfall | |

| 9. Performing Organization Name and Address | 10. Work Unit No. |
|---|---|
| DOT/Transportation Systems Center<br>55 Broadway<br>Cambridge, MA 02142 | FA-03 |
| | 11. Contract or Grant No. |
| | 13. Type of Report and Period Covered: Technical Report |
| 12. Sponsoring Agency Name and Address<br>Federal Aviation Administration<br>800 Independence Ave., S.W.<br>Washington, D.C. 20546 | 14. Sponsoring Agency Code: FAA RD-123 |

15. Supplementary Notes

16. Abstract

Computer system performance measurement techniques, tools, and approaches are presented as a foundation for future recommendations regarding the instrumentation of the ARTS ATC data processing subsystem for purposes of measurement and evaluation.

Section 1: Introduces the subject of computer system performance measurement and states objectives.

Section 2: Defines several computer system measurement approaches, describes the event-monitoring and statistical sampling software techniques, and discusses the various phases of a measurement process.

Appendix A: Defines the role of an Executive System in diverse computing environments and its effect on the design of a measurement package, discusses fundamental operational concepts of Executive Systems, and reviews ARTS III in terms of those concepts.

Appendix B: Surveys the state-of-the-art of available simulation languages and packages, summarizes their salient characteristics and provides guidelines for evaluation and selection of a simulation capability.

| 17. Key Words | 18. Distribution Statement |
|---|---|
| Computer Measurement, ARTS III Data Processing, Executive Systems, Simulation | |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 71 | |

Form DOT F 1700.7 (8-69)

# COMPUTER SYSTEMS PERFORMANCE MEASUREMENT TECHNIQUES - I

# COMPUTER SYSTEMS PERFORMANCE MEASUREMENT TECHNIQUES - I

## 1.0 INTRODUCTION AND OBJECTIVES

### 1.1 INTRODUCTION

Experience has shown that real-time computer system development is a long, tedious, and expensive process. Corresponding rule-of-thumb software estimates are on the order of two instructions per day and include a percentage breakdown of the various phases [17] as: analysis and design phases - 40%, coding and auditing phases - 15%, and check-out and test phases - 45%.

Experience has also shown that it is almost impossible to generate software of any complexity which successfully and reliably accomplishes its intended purpose and at the same time with no extra effort also achieves a high level of system performance.

Yet prior to a few years ago little attempt was made to evaluate the software systems developed and even as this report is being written, to our knowledge, the art of computer systems performance measurement has not been applied to meet the requirements and needs of a real-time multiprocessor system such as the ARTS ATC System.

Computer systems performance measurement as applicable to the ARTS ATC System, the subject of this report, entails the instrumentation of that System for purposes of determining specifically what is going on while the system is executing. The system developers and system evaluators are to be provided the kind of data needed for sound judgments of performance evaluation.

The overall report contains five sections and two appendices in two volumes, of which the present Volume I containing Sections 1 and 2 and Appendices A, and B is the result of FY71 effort. Volume II containing Sections 3, 4 and 5 is expected to appear during FY72. In the present Volume, following an introduction and statement of objectives, Section 2 entitled "Computer Systems Performance Measurement Methodology" begins with a discussion of direct computer measurements.[1] The Section then describes the two main measurement techniques (event-driven and sampling) as well as a number of measurement tools. Lastly, it describes the measure-



[1]This report does not discuss simulation and analytical studies. Schwetman [21] contains a good discussion of these measurement classes.

ment process, paying particular attention to the data reduction and analysis phase.

The appendices contain background information to aid in understanding the problem. Appendix A entitled, "Executive System Fundamentals" describes the overall purpose of Executive Systems and their operational concepts so that the reader may gain better insight into the problem of performance measurement as it relates to Executive operation and total computer system efficiency. Appendix B describes most of the widely used general-purpose simulation languages and system simulation packages. Criteria for evaluating the languages and packages are suggested and few guidelines are given. This completes Volume I.

Volume II will begin with Section 3 describing current performance packages and hardware monitors. Section 4 entitled, "Examination of Promising Performance Measurement Tools" will categorize measurement tools as to their current use and also as to their possible ATC applicability. It will enumerate the attributes and features of measurement tools per se and relate these tools as well as other promising or innovative techniques to the needs and requirements of real-time multiprocessor ARTS ATC Systems. In addition, promising measurement techniques where required are to be evaluated by means of prototype implementation and/or simulation. Section 5 contains the rationale and criteria used for the selection of performance measurement tools, after which the recommended real-time ARTS ATC multiprocessor performance measurement tools are specified.

## 1.2 OBJECTIVES

The principal objectives of this study are:

- Define appropriate measurement tools to monitor and record system activity for purposes of identifying where the time goes in the actual operation of real-time multiprocessing ATC systems.
- Establish hardware/software requirements and specifications for the application of such measurement tools.
- Define the appropriate roles of such performance measurement monitors in the various phases of development of ATC software systems.

## 2.0 COMPUTER SYSTEMS PERFORMANCE MEASUREMENT METHODOLOGY

### 2.1 DIRECT COMPUTER MEASUREMENTS

Direct computer measurement entails the obtaining of measurements of an object computer-controlled system while that system is executing. To date, the means to obtain such data include Hardware Monitors, Software Monitors, and Hybrid Monitors.

2.1.1 Hardware Monitors. Hardware monitors [19] are usually a collection of counters, timers, and electronic probes. These probes or sensors are attached to specific signal wires to measure the presence or absence of electrical impulses. These signals are typically routed into a programmable plugboard, in which wired logic can be used to produce data or information on combinations of functions, such as total I/O time or IO-CPU overlap. Such information obtained through the sensors is summarized and usually written onto magnetic tape for future data reduction. This measurement information can include:

CPU Utilization
I/O Utilization
Peripheral Utilization
Overlap of CPU-I/O
Allocation of Time Between Problem Program and Supervisor
Op Code Usage
Data Base Activity

The advantages of hardware monitors are that they in no way affect or degrade the operation of the system, they can monitor some details of the activity of peripheral equipment that are not reflected in the CPU, they can obtain op code usage, and they can obtain operand statistics. Their disadvantages include their cost, their inability to accurately relate hardware utilization to the software in operation at that time, and they are unable to obtain software tables such as queues, pointers, and program ID's.

2.1.2 Software Monitors. Software monitors for application to standard batch processing and time-sharing computer installations fall into two basic categories [20]:

(1) Configuration-Performance Evaluators
(2) Program-Performance Evaluators

Configuration-Performance Evaluators are designed to monitor the overall hardware performance of the computer installation and its associated peripheral equipment as it processes its normal mix of jobs. It performs this function by measuring the execution activity of all channels and devices during program runs. This measurement information includes:

CPU Utilization
I/O Channel Utilization
Peripheral Utilization
Overlap of CPU-I/O
Queue Length

Program-Performance Evaluators are designed to monitor the performance of program modules by measuring their execution activity. This measurement information includes :

Channel Activity
Program Module Analysis
Entry Point Analysis (time spent in each subroutine by entry point)
Timing Graph
Direct Time Distribution

Software monitors can provide comprehensive cause and effect analysis of system performance. They can obtain quantitative variables which are concerned with the magnitude of some quantity and descriptive variables which identify the respective system element or, in the case of program-performance, the module and segment names of the program being measured. The level of data and the amount of data can be varied to meet the measurement needs of the system. The disadvantages of software monitors are that they incur additional overhead execution time which in turn distorts the system. However, typically the overhead time incurred is 1-5% depending on the measurement techniques and their frequencies of usage. Other disadvantages of software monitors are their prohibitive execution time in obtaining op code usage and operand statistics.

2.1.3 Hybrid Monitors. Hybrid monitors utilize both hardware and software measuring tools to gather pertinent data, together with a software analysis capability resident in an on-line auxiliary computer. The auxiliary computer processes these data in a variety of ways:

1. In real-time, through the use of displays, as a quick-look type system performance monitor.

2. Operating hardware measuring tools by means of program control.

3. Initiate software measuring tools as a result of the occurrence of selected hardware monitor events.

4. Relating the hardware obtained data to the software in operation at the time.

5. As a dynamic system performance monitor.

6. Formatting and recording the data obtained by hardware and software for future data reduction and analysis.

## 2.2 MEASUREMENT TECHNIQUES AND TOOLS

2.2.1 Measurement Techniques. There are two types of measurement techniques: event-driven techniques and sampling techniques. A description of each follows.

2.2.1.1 Event-Driven. An event-driven measurement technique is utilized to record the occurrence of selected events within a computer system. Hardware monitors are in this class of technique as are many software measurement tools.

The mechanism for a software event-driven technique is a "hook" which is strategically implanted at various points throughout the software (both in the supervisor and the user's program modules). This mechanism generally consists of two instructions (e.g., EX SWITCH and NOP HOOKID CODE). When control reaches the hook, two possibilities exist:

1. During a recording run, the hook causes control to transfer to the software monitor. The monitor in turn inspects the second instruction of the hook, thereby obtaining the identifying code which is used to index to the appropriate collector routine. Control subsequently returns to the instruction just past the hook with the machine registers undisturbed.

2. During a normal run, control passes through the hook, i.e., it functions as a pair of NOP instructions.

For IBM 360 series computers, this technique is further refined by the use of a supervisor call instruction (SVC) that allows a program to cause a unique, identifiable hardware interrupt. Implicit in the hardware processing is the saving of the address of the next executable instruction, which permits a return to be effected without requiring the use of a working register for linkage. This SVC instruction, for the recording

run, was previously stored into SWITCH and when control arrives at a hook, it is this SWITCH that is the subject of an EXECUTE instruction. In addition, all other hardware interrupts (except machine checks) are transferred to the software monitor with the CPU being placed in the supervisor state (i.e., all interrupts marked). These interrupts are then viewed as belonging to the following categories:

- o Normal interrupts: The occurrence is time stamped, logged in the output buffer, and the interrupt is passed on to its appropriate routine.

- o Software monitor interrupts: Each I/O interrupt is checked to see if it belongs to the software monitor, i.e., resulted from a software monitor initiated tape operation. If not, it is processed as a normal interrupt. If it does belong to the software monitor, the completion is noted (buffer housekeeping) and control is returned to the system at the point of interrupt.

- o Hook interrupts: Each interrupt is checked to see if it was the result of executing a hook. If not, it is processed as a normal interrupt. Hook processing is terminated by returning control to the interrupted module at a point immediately after the hook location.

Software event-driven technique provides the means of obtaining the most detailed insight of the system in operation. This presupposes a good deal of knowledge of the system on the part of the user to judiciously select the strategic locations for the hooks as well as the appropriate data to be recorded by their respective collector type. The amount of data to obtain in any given run depends on the needs of the user and also on the degradation that the system can tolerate. This problem may partially be solved by implementing a selectivity option such that as a function of parameters, entered at the start of a run the monitor can "turn off" any hook or group of hooks for the duration of that run. Also the number and type of hooks depends on the stage of the development of the system per se.

2.2.1.2 Sampling. Sampling technique involves interrupting the application program periodically and recording status information. The obvious advantage of this procedure is that the time spent gathering data is only a fraction of that required to completely trace the execution of the program. Currently available software packages utilizing sampling incur a 1 to 5% incrase in overhead.

2.2.1.2.1 A Statistical Model of the Sampling Process. Assume that an application program is executing. Nothing is known about the language or logic of the program. Periodically the program is interrupted and the address of the next instruction to be executed is recorded. No assumptions can be made about the process being sampled, but the random sampling procedure has definite characteristics--an intersample time or sampling interval and a sample size.

The objective of the sampling procedure is to estimate the time distribution for the execution of the program over all subroutines and address locations. The best estimator for the percentage of time spent at a given location is the number of times the location was observed divided by the total number of samples taken.

The sampling procedure must be designed to satisfy a given level of accuracy or confidence limit. Two parameters of the sampling procedure can be specified to control the accuracy of the results; they are the sample size and the sampling interval.

2.2.1.2.2 Sample Size. In order to derive limits for the proportions which are observed, a distribution free statistical test must be used since nothing is known about the program to be sampled. Bradley [1] suggests the following procedure:

Let n = sample size
r = observed # at the location
p = true % time spent at the location
$p_1$ = lower confidence limit for p
$p_2$ = upper confidence limit for p
$\alpha$ = 1-confidence level
$\alpha_1 = \alpha_2 = \alpha/2$

The confidence level selected by the experimenter represents the probability that the true percentage is less than $p_2$ but greater than $p_1$. If the observed result falls within these limits it can be accepted as a valid estimate for the percentage of time spent at the location.

Using the binomial distribution the following equations are derived:

$$\alpha_1 = \sum_{i=r}^{n} \binom{n}{i} p_1^{\,i}(1-p_1)^{n-i}$$

$$\alpha_2 = \sum_{i=0}^{r} \binom{n}{i} p_2^{\,i}(1-p_2)^{n-i}$$

These equations can be solved for $p_1$ and $p_2$ by entering the binomial tables with n and r and finding the largest p satisfying the first equation and the smallest p satisfying the second. Then at confidence level $1-\alpha_1-\alpha_2, p_1<p<p_2$. The confidence interval can be calculated for different values of n and r to compare the increased accuracy with increased sampling.

For large values of n (greater than 100) r/n converges to p. Binomial tables usually do not contain values for n greater than 30 so it is convenient to use another approximating technique for calculating the confidence interval. The random variable:

$$Z = \frac{r-np}{\sqrt{n(r/n)(1-r/n)}}$$

(see Hogg and Craig, [3] is distributed n(0,1). Using the standard normal tables we can find the confidence interval for given confidence levels. For example, if n = 1000 and we observed r = 100, then $.081<p<.119$ with probability .05 and the estimate r/n = .10 may be used as a valid estimate of p.

Application. A family of confidence curves can be derived using the theory described above [20] (see Figure 2.1). Each curve represents a different confidence level or a different probability that the results will be within the interval read from the graph. For example, if the sample size is 8000, the observed frequency values will fall within 2 percent of the actual value with probability .9999. Also, for a sample of 8000, the probability that the results are within 1 percent of the actual value is .95. The confidence curves can be used to determine the appropriate sample size for a particular confidence level and degree of accuracy as well as checking the accuracy of a particular sample size and confidence level.

2.2.1.2.3 Sampling Interval. The timing of the sampling interrupts may be periodic or random. Periodic sampling uses a fixed intersample time which is calculated by dividing the expected run time of the program, T, by the desired number of samples, n. Boole and Babbage's PPE[4] uses this method.

Random sampling does not use a fixed sampling interval. A function for calculating the intervals is part of the measurement program. Lambda's LEAP[4] uses a random sampling technique.

[4] See Section 3 of Volume II for a description of these commercially available proprietary packages.

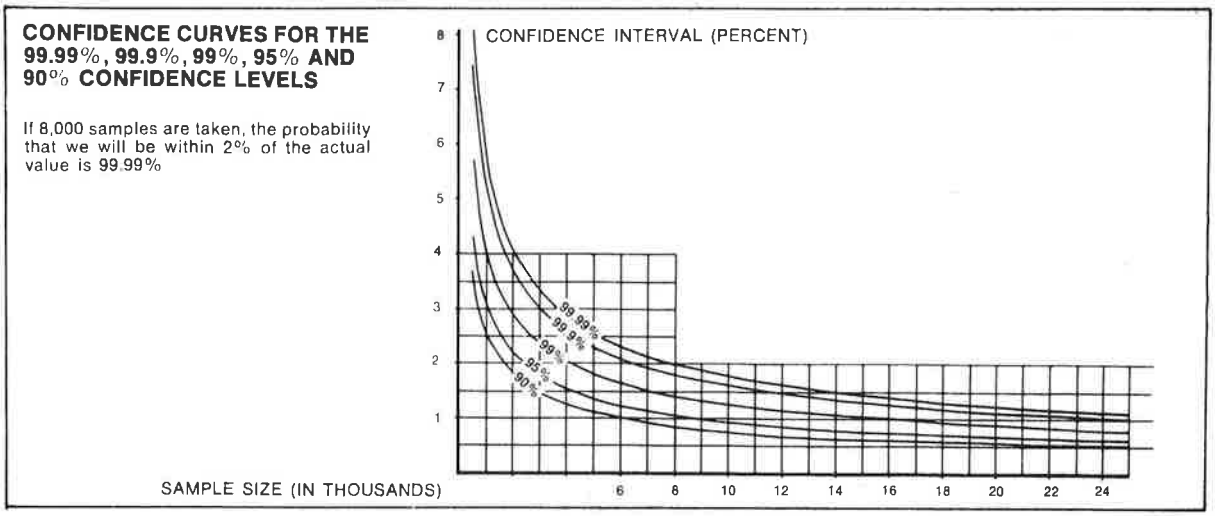



FIGURE 2.1
ACCURACY OF DATA AS A FUNCTION OF A NUMBER OF SAMPLES

Calculation of the Random Sample Interval. Assume that the probability that a sample will occur at time t during the duration of the measurement is distributed according to the Poisson function. Then the intersample times are described by the negative exponential function:

$$p(t) = \lambda e^{-\lambda t}$$

where $\lambda$ is the mean number of samples and $1/\lambda$ is the mean intersample time. After each sample is taken the next sample is scheduled using a random number generator and the above expression. The total number of samples can be controlled by adjusting $\lambda$, before or during execution of the application program. (If $\lambda$ were changed during the sampling procedure, then the weight of the samples would have to be adjusted accordingly.) If the approximate execution time is known, $\lambda$ can then be set to $n/T$ (or $1/\lambda = T/n$). Note that for periodic sampling the interval is set to $T/n$ while random sampling uses an interval which varies about this value.

Comparison of Periodic and Random Sampling. There are differing opinions as to which method of interval selection is better. If periodic sampling is used and the sampling period is in synchronization with a set of instructions, then the results will not reflect the true time distribution over the entire program. Proponents of the fixed sampling interval claim that the analyst interpreting the results of the measurement would be able to spot synchronization and could alter the sampling interval accordingly. Cantrell and Ellison [5] do not find possible synchronization a problem and dismiss the issue.

2.2.2 Measurement Tools.[2] Sophisticated tools are needed to measure what is going on inside a system while the system is executing. Such tools in their implementation and/or fabrication utilize either the event-driven or the sampling measurement technique together with software, or hardware, or both. These tools are encompassed in an appropriate monitor and should inherently relate to the system being measured, reflecting measurement know-how and ingenuity.

A partial list of hardware, software, and hybrid tools follows.

2.2.2.1 Hardware Tools

2.2.2.1.1 Calendar Clock. A program readable calendar clock with an associated program loadable time match register that is continuously being compared by means of independent hardware for purposes of generating an interrupt whenever a time match occurs.

This hardware feature enables one to utilize the measurement sampling technique. Typical application, which is initiated by the occurrence of an interrupt, might be the capturing and recording of pertinent register data and program module type activity information.

2.2.2.1.2 Interval Timer. A program loadable time register which generates an interrupt at the end of the appropriate time interval.

This hardware feature, although not as accurate timewise as the Calendar Clock, functions in a similar manner.

2.2.2.1.3 Memory Cycle Counter. The memory cycle counter counts the number of memory references made by the central processor.

Typical application would be the obtaining and recording of the contents of this counter either periodically or at the completion of a run.

2.2.2.2 Software Tools

2.2.2.2.1 Supervisor Module Metering Package. This tool records time spent executing selectable supervisor modules. For each selected module the metering package records the number of times the module is invoked and the total execution time accumulated within each of a number of ranges of execution times for the respective module.

The tool in its implementation utilizes the event-driven measurement technique.

---

[2] Saltzer and Gintell [15] presents an excellent discussion of measurement tools used in the instrumentation of Multics.

2.2.2.2.2 Segment Utilization Metering Facility. A segmented system provides a simple way to detect how time spent in the system is distributed among the various components. This tool, utilizing the sampling technique, sets the calendar clock to interrupt periodically. When the interrupt occurs, the segment number of the segment which was executing is used to index into an array of per-segment counters and the appropriate counter is incremented by one. After the system has run for a while the resulting distribution of segment utilization can be outputted.

2.2.2.2.3 Entries/Exits Facility. An "add-one-to-storage" instruction is included in this sequence which increments a counter each time a procedure or routine is entered. These counters enable a programmer to determine later how many times a procedure has been called and to relate that number to the number of calls to other procedures.

2.2.2.2.4 Missing Page Tracing Package. This package is used in a virtual memory system and retains in a ring buffer the segment, page number, and the time of day of the last 256 missing pages of the process under measurement. Thus the data obtained provides the programmer with sufficient information on how to reorganize his program to improve its locality of reference.

2.2.2.2.5 Scheduling Algorithm Tracing Package. The general strategy here is to write a user program which goes into a tight loop repeatedly reading the calendar clock. Normally, successive clock readings differ by the loop transit time. If a larger difference occurs, it is the result of the processor handling an interrupt or executing another process. These larger time differences, as well as the time they were noted, are entered in a table and control returns to the aforementioned loop. When the table is filled, its contents is written onto magnetic tape for subsequent processing. This information helps build confidence that the processor scheduling algorithm is working as predicted and it also provides an independent confirmation of the time required to handle each interrupt.

2.2.2.3 Hybrid Tools

2.2.2.3.1 Externally-Driven I/O Channel. This hybrid feature, an externally-driven input/output channel, currently utilized in the instrumentation of Multics, permits another computer to monitor the contents of primary memory. The channel is connected by a 2400 baud telephone line to a PDP-8/338 programmable display computer. The data rate involved--less than 60 words per second--presents a negligible I/O and memory cycle load to the Multics computer.

2.2.2.3.2 Graphic Display Monitor. The Graphic Display Monitor is a subsystem of PDP-8/338 programs that use the aforementioned externally-driven I/O channel to interrogate locations of Multics memory. Multics in turn during system initialization generates a table containing pointers

to interesting data bases. Standard displays have been developed to observe queues, the arrays of module execution time distributions, and the use of primary memory. Reportedly, observations of these displays have been helpful in detecting bottlenecks in the system, and on several occasions have exhibited the system passing through states previously thought to be impossible.

## 2.3 MEASUREMENT PROCESS

The measurement process for computer system performance evaluation entails data capturing, data collection, and data reduction and analysis.

2.3.1 Data Capturing. This is achieved either by hardware instrumentation, software instrumentation, or hybrid instrumentation. As previously stated, when utilizing the event-driven measurement technique, proper selection of the events to be monitored are of prime importance.

2.3.2 Data Collection. Generally magnetic tapes are used to record the data. For the sampling measurement technique each sample might include such data as an absolute instruction address, the name of the program module, its segment number if the module is using overlays, and whether the module was waiting (i.e., it is unable to proceed until the occurrence of a known event such as I/O completion) or executing. Similarly, the event-driven measurement technique's data might include the program module or event name, a time stamp, unique data associated with the event such as queue lengths, queues, tables, pointers, variables, etc.

The key to high effectiveness in a measurement tool involves the prior identification of the types of data most likely to be useful in the performance evaluation of the object system (i.e., data collection phase) and the tailoring of the information developed by the data reduction and analysis phase to meet the needs of the system developers and the system evaluators.

2.3.3 Data Reduction and Analysis. This function is generally performed off-line, encompassing a good deal of capability and flexibility in the reducing of data collection tape(s). As previously stated, the information developed by this phase should be tailored to meet the needs of the system developers and the system evaluators.

The information per se should constitute comprehensive cause and effect analysis of system behavior. Further, the information should be presented depicting both a broad understanding of the system's overall timing and a detailed time distribution of preselected program modules or system components.

For illustrative purposes, a description of data reduction and analysis type reports follows.[3]

2.3.3.1 Channel Activity. The channel activity summary reports the activity of all data sets encountered in the run. Statistics are printed on the open, busy, and wait status of each data set, as well as on the control unit busy status.

CHANNEL ACTIVITY SUMMARY

DDNAMES ENCOUNTERED IN 6.68 MINS.
SCANNED FOR I/O ACTIVITY: 5

TOTAL I/O WAIT TIME 0.0 MINS. (= 0.0 %).
AT LEAST ONE DATA SET OPEN FOR 6.68 MINS. (=100.00 %).
AT LEAST ONE DATA SET BUSY FOR 0.00 MINS. (= 0.05 %).
AT LEAST ONE CONTROL UNIT BUSY FOR 6.68 MINS. (=100.00 %).

| DDNAME | FUNCTION | DEVICE | UNIT | OPEN (MINS., %) | | BUSY (MINS., %) | | WAIT (MINS., %) | | UNIT (MINS., %) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LAMERROR | WRIT | DISK- 8 | 246 | 6.68 | 100.00 | 0.00 | 0.05 | 0.0 | 0.0 | 6.68 | 100.00 |
| MRESULTS | WRIT | TAPE- 1 | 0C2 | 6.68 | 100.00 | 0.0 | 0.0 | 0.0 | 0.0 | 6.68 | 100.00 |
| FT05F001 | READ WRIT | DISK- 8 | 337 | 6.68 | 100.00 | 0.0 | 0.0 | 0.0 | 0.0 | 6.68 | 100.00 |
| PROLOG | EXCP | TAPE- 1 | 0C1 | 6.68 | 100.00 | 0.0 | 0.0 | 0.0 | 0.0 | 6.68 | 100.00 |
| FT06F001 | READ WRIT | DISK- 8 | 247 | 6.68 | 100.00 | 0.0 | 0.0 | 0.0 | 0.0 | 6.68 | 100.00 |

CPU ACTIVE IN I/O PROCESSING FOR 0.00 MINS. (= 0.05 % OF 6.68 MINS. SCANNED)

2.3.3.2 Module Analysis. The module analysis identifies by name and function the modules involved in the execution of the program and reports the space and time usage of each.

MODULE ANALYSIS

ENCOUNTERED IN 6.68 MINS.
MODULES : 8
SVC ID'S: 0

| MODULE NAME | | ORIGIN | LENGTH | ENTRY | OVERLAY REG | OVERLAY SEG | OVERLAY LNK | TIME (MINS.) DIRECT | TIME (MINS.) INDIRECT |
|---|---|---|---|---|---|---|---|---|---|
| MAIN | | 000C00 | 008FC8 | 06E860 | | | | 6.68 (=100.00 % ) | 6.68 |
| | | 000C00 | 008FC8 | | 1 | 1 | 0 | 4.87 (= 72.90 % THIS MODULE) | |
| | | 040700 | 06E838 | | 1 | 2 | 1 | 0.0 (= 0.0 % THIS MODULE) | |
| | | 070400 | 06E838 | | 1 | 3 | 1 | 0.0 (= 0.0 % THIS MODULE) | |
| | | 070700 | 021700 | | 1 | 4 | 1 | 1.80 (= 26.90 % THIS MODULE) | |
| IGG019CD | OPENCLOS | 0FE960 | 000208 | 0FE960 | | | | 0.0 (= 0.0 % ) | 0.0 |
| IGG019CC | OPENCLOS | 0FE908 | 000058 | 0FE908 | | | | 0.0 (= 0.0 % ) | 0.0 |
| IGG019CI | OPENCLOS | 0FE820 | 000078 | 0FE820 | | | | 0.0 (= 0.0 % ) | 0.0 |
| IGG019CH | OPENCLOS | 0FE898 | 000070 | 0FE898 | | | | 0.0 (= 0.0 % ) | 0.0 |
| IGG019BA | OPENCLOS | 0FE618 | 000180 | 0FE618 | | | | 0.0 (= 0.0 % ) | 0.0 |
| IGG019BB | OPENCLOS | 0FE5C0 | 000058 | 0FE5C0 | | | | 0.0 (= 0.0 % ) | 0.0 |
| IEWSZOVR | LOADER | 078CC8 | 000338 | 078CC8 | | | | 0.0 (= 0.0 % ) | 0.0 |
| ENTIRE PARTITION (REGION) | | 06E800 | 00F000 = 60 K BYTES | | | | | | |
| TOTAL FREE (BLOCK) SPACE | | | 003800 = 14 K BYTES | | | | | | |

CPU PROCESSED SVC CALLS FOR 0.0 MINS. (= 0.0 %).
PRB EXECUTED FROM PROTECTED CORE FOR 0.0 MINS. (= 0.0 %).

2.3.3.3 Entry Point Analysis. This report, entry point analysis, shows the time spent in each subroutine by entry point. This includes the direct time, the indirect time (time spent in lower level calls through the subroutine), and the total of these two. The indirect time is further pinned down: For each subprogram, the report shows the subprograms it calls (and the relative time among these). For convenience, the entry point analysis is ordered in three ways, as can be seen from the example on this sheet. In summary, this analysis tells at a glance "where the action is" and "who" caused it at the level of the logic of subprogram flow.

ALPHABETIC ORDER

ORDERED BY DIRECT TIME

ENTRY POINT ANALYSIS

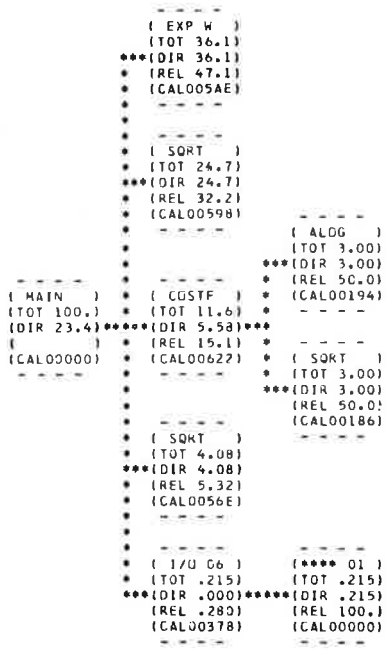
ORDERED BY TOTAL TIME

| ENTRY | PERCENT DIRECT TIME | PERCENT INDIRECT TIME | PERCENT TOTAL TIME | CALLS ON: ENTRY | CALLS ON: RELATIVE PERCENT | CALLED BY: ENTRY | CALLED BY: RELATIVE PERCENT |
|---|---|---|---|---|---|---|---|
| MAIN | 23.4 | 76.6 | 100.0 | EXP W<br>SQRT<br>COSTF<br>I/O 06 | 47.1<br>37.5<br>15.1<br>0.3 | | |
| EXP W | 36.1 | 0.0 | 36.1 | | | MAIN | 100.0 |
| SQRT | 31.8 | 0.0 | 31.8 | | | MAIN<br>COSTF | 90.5<br>9.5 |
| COSTF | 5.6 | 6.0 | 11.6 | ALOG<br>SQRT | 50.0<br>50.0 | MAIN | 100.0 |
| ALOG | 3.0 | 0.0 | 3.0 | | | COSTF | 100.0 |
| I/O 06 | 0.0 | 0.2 | 0.2 | **** 01 | 100.0 | MAIN | 100.0 |
| **** 01 | 0.2 | 0.0 | 0.2 | | | I/O 06 | 100.0 |

| ENTRY |
|---|
| ALOG |
| COSTF |
| EXP W |
| I/O 06 |
| MAIN |
| SQRT |
| **** 01 |

| ENTRY |
|---|
| EXP W |
| SQRT |
| MAIN |
| COSTF |
| ALOG |
| **** 01 |
| I/O 06 |

2.3.3.4 Timing Graph. The entry point call structure timing graph is a tree-like diagram of the significant paths through the hierarchy of subprogram calls from the main program to final direct execution. This information allows the user to identify the individual most significant call chains and their relative timing. A given subprogram may make calls on another subprogram from more than one place, and it is therefore important to be able to understand the timing implications among these separate calls.




2.3.3.5 Direct Time Distribution. The direct time distribution analysis shows at a glance the detailed distribution of running-time spent by individual instruction locations in core, at the machine code level. For programs in higher-level languages, this analysis is related to the original source program by reference to assembly listings. For each interval of code, the report reflects absolute time, percent time, and time density. The graphic display of time density is no mere histogram, as the intervals are adjusted automatically to magnify trouble spots for easier analysis. Thus, one often obtains resolution to the level of individual machine instructions, but only when that level is significant.

## DIRECT TIME DISTRIBUTION ANALYSIS

| ENTRY | ABSOLUTE INTERVAL | RELATIVE INTERVAL | PERCENT TIME | CUMULATIVE PERCENT TIME | TIME DENSITY | INDICATOR |
|---|---|---|---|---|---|---|
| | ROUT SEGMENT - SYSTEM AND LIBRARY ROUTINES | | | | | |
| | 6F422-6F423 | 6F422-6F423 | 0.0 | 0.0 | 0.0 | |
| ALOG | 6F424-6F45F | 00000-0003B | 0.9 | 0.9 | 0.014 | |
| | 6F460-6F47F | 0003C-0005B | 1.3 | 2.1 | 0.040 | * |
| | 6F480-6F5C7 | 0005C-001A3 | 1.3 | 3.4 | 0.004 | |
| EXP W | 6F5C8-6F5D3 | 00000-00008 | 0.4 | 3.9 | 0.036 | * |
| | 6F5D4-6F5DB | 0000C-00013 | 1.1 | 4.9 | 0.134 | ** |
| | 6F5DC-6F5E3 | 00014-0001B | 1.3 | 6.2 | 0.161 | *** |
| | 6F5E4-6F5FB | 0001C-00033 | 1.3 | 7.5 | 0.054 | * |
| | 6F5FC-6F5FF | 00034-00037 | 0.6 | 8.2 | 0.161 | *** |
| | 6F600-6F603 | 00038-0003B | 1.3 | 9.4 | 0.322 | ****** |
| | 6F604-6F611 | 0003C-00049 | 1.1 | 10.5 | 0.077 | * |
| | 6F612-6F619 | 0004A-00051 | 1.1 | 11.6 | 0.134 | ** |
| | 6F61A-6F61D | 00052-00055 | 1.5 | 13.1 | 0.376 | ******* |
| | 6F61E-6F621 | 00056-00059 | 1.3 | 14.4 | 0.322 | ****** |
| | 6F622-6F631 | 0005A-00069 | 1.1 | 15.5 | 0.067 | * |
| | 6F632-6F635 | 0006A-0006D | 1.5 | 17.0 | 0.376 | ******* |
| | 6F636-6F641 | 0006E-00079 | 1.3 | 18.2 | 0.107 | ** |
| | 6F642-6F643 | 0007A-0007B | 1.9 | 20.2 | 0.966 | ****************** |
| | 6F644-6F647 | 0007C-0007F | 0.4 | 20.6 | 0.107 | ** |
| | 6F648-6F649 | 00080-00081 | 2.1 | 22.7 | 1.073 | ******************** |
| | 6F64A-6F653 | 00082-0008B | 1.5 | 24.2 | 0.150 | *** |
| | 6F654-6F655 | 0008C-0008D | 3.4 | 27.7 | 1.717 | ********************************* |
| | 6F656-6F657 | 0008E-0008F | 0.2 | 27.9 | 0.107 | ** |
| | 6F658-6F65F | 00090-00097 | 1.7 | 29.6 | 0.215 | **** |
| | 6F660-6F667 | 00098-0009F | 1.5 | 31.1 | 0.188 | *** |
| | 6F668-6F669 | 000A0-000A1 | 4.5 | 35.6 | 2.253 | ********************************************** |
| | 6F66A-6F67F | 000A2-00087 | 1.3 | 36.9 | 0.059 | * |
| | 6F680-6F68D | 00088-000C5 | 1.5 | 38.4 | 0.107 | ** |
| | 6F68E-6F69D | 000C6-00005 | 1.5 | 39.9 | 0.094 | ** |
| | 6F69E-6F77D | 00006-00185 | 0.6 | 40.6 | 0.003 | |
| I/O 06 | 6F77E-70C3F | 00000-014C1 | 0.0 | 40.6 | 0.0 | |
| SQRT | 70C40-70C47 | 00000-00007 | 0.9 | 41.4 | 0.107 | ** |
| | 70C48-70C4B | 00008-0000B | 1.1 | 42.5 | 0.268 | ***** |
| | 70C4C-70C4F | 0000C-0000F | 1.3 | 43.8 | 0.322 | ****** |
| | 70C50-70C57 | 00010-00017 | 1.3 | 45.1 | 0.161 | *** |
| | 70C58-70C6B | 00018-0002B | 1.7 | 46.8 | 0.086 | ** |
| | 70C6C-70C6D | 00C2C-00020 | 2.6 | 49.4 | 1.288 | ************************ |
| | 70C6E-70C6F | 0002E-0002F | 0.0 | 49.4 | 0.0 | |
| | 70C70-70C71 | 00030-00031 | 1.9 | 51.3 | 0.966 | ****************** |
| | 70C72-70C7F | 00032-0003F | 1.3 | 52.6 | 0.092 | ** |
| | 70C80-70C83 | 00040-00043 | 1.3 | 53.9 | 0.322 | ****** |
| | 70C84-70C89 | 00044-00049 | 1.7 | 55.6 | 0.286 | ***** |
| | 70C8A-70C8B | 0004A-0004B | 5.2 | 60.7 | 2.575 | ************************************************** |
| | 70C8C-70C99 | 0004C-00059 | 1.3 | 62.0 | 0.092 | ** |
| | 70C9A-70C9B | 0005A-0005B | 3.4 | 65.5 | 1.717 | ********************************* |
| | 70C9C-70CA1 | 0005C-00061 | 1.5 | 67.0 | 0.250 | **** |
| | 70CA2-70CA3 | 00062-00063 | 5.6 | 72.5 | 2.790 | ****************************************************** |
| | 70CA4-70CA5 | 00064-00065 | 0.9 | 73.4 | 0.429 | ******** |
| | 70CA6-71555 | 00066-00915 | 1.3 | 74.7 | 0.001 | |
| **** 01 | 71556-7671F | 00000-051C9 | 0.2 | 74.9 | 0.000 | |
| | OVERLAY FOUR - START OF PROGRAM CODE | | | | | |
| MAIN | 76720-76C5F | 00000-0053F | 0.0 | 74.9 | 0.0 | |
| | 76C60-76C7F | 00540-0055F | 1.5 | 76.4 | 0.047 | * |
| | 76C80-76CAD | 00560-0058D | 1.1 | 77.5 | 0.023 | |
| | 76CAE-76CBD | 0058E-0059D | 1.7 | 79.2 | 0.107 | ** |
| | 76CBE-76CBF | 0059E-0059F | 3.4 | 82.6 | 1.717 | ********************************* |
| | 76CC0-76CC5 | 005A0-005A5 | 0.9 | 83.5 | 0.143 | *** |
| | 76CC6-76CD5 | 005A6-00585 | 1.7 | 85.2 | 0.107 | ** |
| | 76CD6-76CDB | 00586-0058B | 0.9 | 86.1 | 0.143 | *** |
| | 76CDC-76CE9 | 0058C-005C9 | 1.3 | 87.3 | 0.092 | ** |
| | 76CEA-76CEB | 005CA-005CB | 3.2 | 90.6 | 1.609 | ******************************** |
| | 76CEC-76CF1 | 005CC-005D1 | 1.3 | 91.8 | 0.215 | **** |
| | 76CF2-76CF3 | 005D2-005D3 | 0.6 | 92.5 | 0.322 | ****** |
| | 76CF4-76D19 | 005D4-005F9 | 1.3 | 93.8 | 0.034 | * |
| | 76D1A-76D57 | 005FA-00637 | 1.3 | 95.1 | 0.021 | |
| | 76D58-76E57 | 00638-00737 | 0.2 | 95.3 | 0.001 | |
| COSTF | 76E58-76F45 | 00000-000ED | 1.1 | 96.4 | 0.005 | |
| | 76F46-76F83 | 000EE-0012B | 1.3 | 97.6 | 0.021 | |
| | 76F84-76FA1 | 0012C-00149 | 1.5 | 99.1 | 0.050 | * |
| | 76FA2-76FC0 | 0014A-00168 | 0.9 | 100.0 | 0.028 | |

# APPENDIX A

## EXECUTIVE SYSTEM FUNDAMENTALS

## CONTENTS

# APPENDIX A

# EXECUTIVE SYSTEM FUNDAMENTALS

## 1.0 INTRODUCTION

In order to gain insight into the problem of performance measurement as it relates to Executive operation and total computer system efficiency, one must have an understanding of the overall purpose of Executive Systems and their operational concepts. Proper measurement and intelligent evaluation of a particular system requires knowledge of the specific design goals of that system together with its implementation details.

The design goals and rationale for implementation decisions in the ARTS III system can be better understood and highlighted by a background in general Executive System fundamentals. This Appendix attempts to define the role of an Executive and to introduce some of the basic characteristics which distinguish the design philosophies of General-Purpose Batch Processing, General-Purpose Time Sharing, and Real Time Executive Systems for the purpose of demonstrating the comparative role of various types of Executive in different computer environments. Fundamental concepts indigenous to the operation of most large-scale Executive systems will be presented. Finally, the ARTS III System will be reviewed in terms of its design classification and operational concepts.

## 2.0 THE ROLE OF AN EXECUTIVE SYSTEM

An Executive System, broadly stated, is a collection of special-purpose programmed processes which operate in, control, and service a computer facility.

Its major responsibilities usually include:

1. Scheduling and monitoring the flow of programs and data.
2. Management of hardware and software resources.
3. Synchronization and control of input/output operations on behalf of applications programs.
4. Protection of programs and data from unauthorized access or destruction by other programs.
5. Communication between the internal activity of the computer system and the operator or programmer.
6. Participation in hardware and software error detection, diagnosis, and recovery.

Basically, an Executive System provides services which are performed in response to requests by "users". A user may be an operator, a program(mer),

or a piece of hardware such as a teletypewriter, remote terminal, or central processor. The extent to which any of these services are supplied, as well as the algorithms and techniques employed in supplying them, may vary greatly from installation to installation. Hardware design vagaries and diversified computing requirements account for dissimilarity among Executive Systems.

## 3.0 SYSTEM PROFILES AND DESIGN PHILOSOPHIES

The basic design philosophy of a particular Executive is strongly influenced by the primary "type" of service it is intended to supply. The need for a tailored approach to Executive design may best be illustrated by a brief discussion of several distinct computing environments and some of their evincive characteristics.

### 3.1 General-Purpose Batch Processing

Assume the primary type of service to be provided is one of General-Purpose Batch Processing.

#### 3.1.1 Response Time and Man-Machine Interaction

One of the fundamental attributes of any computer system is its "response time". The response time of current Batch Processing systems is referred to as "turnaround time" and is usually measured in minutes or hours. It is defined as the time elapsed between submittal of a job for processing and return of output from that job. The programmer has no contact with the job during that time, hence no programmer-machine interaction. All resource information necessary to process a job must be provided, in advance of execution, by the programmer via some form of control language. This mode of operation is most suitable for production runs where changing data is processed against programs which have been written and debugged; in other words, jobs which are not subject to daily code modification.

#### 3.1.2 Workload Implications

Since the overall mission is General Purpose, the Executive should be designed to manage a wide variety of applications such as simulation, mathematical analyses, data reduction, report generation, list processing, and various scientific and commercial studies. Such disparity of application is usually accompanied by different data-base requirements and gives rise to a need for variety in peripheral equipment, i.e., magnetic tape, random access disk, card readers, line printers, plotters and punches. It follows that support for these devices and data bases must be provided to the programming community in the form of comprehensive I/O (input/output) packages comprising a tractable approach to the

handling of differing record formats, file organizations, data access characteristics and device peculiarities. These considerations contribute complexity and extensive code to the Executive.

### 3.1.3 Scheduling Considerations

Inherent in a General-Purpose system is a wide variation in job running time, memory space requirements, and processor vs channel use. These attributes, combined with an assortment in number and type of computer user, affords a lack of predictability about workload which must be addressed through flexibility in the scheduling process.

Batch Processing describes the manner in which jobs are collected and run. Jobs gathered by operators are physically read into the system in batches, placed in queues, and selected for execution by means of a scheduling algorithm. Customarily, the scheduling algorithm for a General-Purpose Batch Processing situation is designed to deal with the relative priorities of jobs in the queues and their anticipated running time. It should also allow for operator alteration of the priority structure of queued jobs. Its prime orientation within the priority framework is to achieve optimal use of critical machine resources such as core storage, processor time, I/O time, and common auxiliary storage devices through various "load balancing" techniques. The decisions incorporated in the scheduling algorithm often demand that a fairly elaborate statement of the resource requirements for each job be made by the programmer. This is accomplished via control cards which accompany the job when it is submitted for processing. The information coded on control cards is interpreted and stored in tables by the interpreter routine of the Executive for subsequent use by the scheduler and other resource management modules. Control cards are a characteristic means of communication in the Batch Processing System, since there is no programmer interaction during job processing.

### 3.1.4 Overhead

Overhead in a computer system falls into two categories. (1) Time spent executing Executive code or, from another point-of-view, time spent away from application work, and (2) Volume of Executive code.

Since a General-Purpose system is by definition broad in its potential scope of application, it usually requires more overhead time than special-purpose systems in order to properly co-ordinate diverse activity and provide specialized assistance to individual programs. A great deal of complex code is typical of such systems for much the same reasons.

However, the implementation of General-Purpose Executives is for the most part modular; that is, constructed of independent and asynchronous processes. It is, therefore, not necessary for the Executive to be wholly contained in main memory at any one time. The lenient expected response time (turnaround time) in General-Purpose Batch Processing systems makes tolerable the small amount of additional overhead time required to swap certain Executive modules in and out of core storage on an as-needed basis. This procedure makes it possible to leave larger amounts of core storage available to the applications programs while still providing those programs with desirable Executive services.

### 3.1.5 Storage Allocation

In representative General-Purpose Batch Processing systems, a given program is allotted a portion of main memory large enough to contain the complete program (or the largest overlay segment) and its necessary data buffers. That portion of storage is reserved by the program until it terminates execution. The programmer is usually charged with specifying in advance the maximum amount of core storage the job will require. Alternatively, the Executive may be appointed to calculate the program's memory requirement from other available information, or default to a fixed value. In either case, the space a program may occupy is restricted only by the total amount of main storage available in the computer configuration, or local convention. Since there is usually more than one job in main memory at any given time, the memory requirements of a job will more than likely affect the scheduling of that job for execution and resultantly, will impact the response time. A Batch Processing Executive normally attempts to maximize the use of main storage, because it is an expensive commodity. In some systems, it is possible for the programmer to dynamically release portions of the memory originally allotted to his program. When this occurs, the Executive may find sufficient unassigned space to schedule another program thereby improving efficiency. On the other hand, a few systems also provide the ability to request additional memory space during program execution. In this event, it may be necessary for the Executive to remove or "roll-out" a lower priority program in its entirety in order to assign the additional space to the requesting routine. The removed program will be automatically rolled-in again for subsequent completion when storage is available. This procedure gives the programmer latitude in optimizing his own use of memory. Variations on these basic philosophies can be found in most General-Purpose Batch Processing Systems.

## 3.2 GENERAL-PURPOSE TIME-SHARED SYSTEM

A Time-Shared System is one in which a hardware and software complex is shared among many independent on-line users simultaneously. The sharing should be invisible to the user; that is, each one views the system as solely dedicated to his job. Upholding the illusion of a system dedicated to an individual while serving many users concurrently is a primary design goal of a Time-Shared Executive.

### 3.2.1 Response Time and Man-machine Interaction

Time-shared computer users are connected directly with the system by way of telecommunications media such as telephone lines. Their physical means of communication can vary over a large selection of equipment, but consists, typically, of a typewriter terminal and perhaps a cathode ray tube.

The response time with such a configuration may be defined as the time elapsed between the sending of a message by the typewriter terminal operator and receipt of a reply from the computer. This time is expected to be measured in fractional seconds, seconds, or perhaps minutes subject to the nature of the communication. For example, if the message was a request to compile and execute a program previously filed by the user, it is reasonable to expect a few minutes to pass before output arrives at the terminal site. On the other hand, if the terminal is being utilized in a conversational mode, that is, a programmer is in the process of writing a program and each line of the program is being compiled or scanned for syntax errors by the computer as it is completed, a response in terms of no more than a few seconds is paramount. Otherwise, the user will become impatient and the system will have failed to adequately perform its desired function. There is a high degree of man-machine interaction in a time-shared system.

### 3.2.2 Workload Implications

A Time-Shared environment is extremely useful for interactive writing, debugging and modifying of programs, for making on-the-spot inquiries against a data base, as a teaching aid, and in the realm of graphical displays, where altering parameters related to a mathematical or simulation model allows one to immediately witness the effects of the change. In essence, time-sharing is appropriate whenever problem solving or basic service can be improved by immediate and direct dialogue with the computer. It brings the power of a large computer directly to an office or laboratory and places a potent tool in the hands of the user.

A General-Purpose Time-Shared System must be prepared to simultaneously handle a wide range of applications. The selection of different terminal devices available for these applications imposes an additional requirement

for special and extensive programming support on the Executive system. A pliant, uncomplicated Command Language is essential to effect easy communication between user and system. An elaborate user file system must be managed by the Executive with minimum involvement on the part of the user community. Protection of individual files from accidental or unauthorized access is of particular import in a system where many people are sharing data files and programs as well as the computer hardware. Identification of valid users becomes crucial when anyone with proper terminal equipment may dial into a system over phone lines and potentially usurp time and misuse information. Data arriving concurrently from numerous remote sites must be reliably handled and error-checked. Hardware resources must be dexterously switched from user to user to meet stringent response-time demands. These are characteristic problems to be considered in Time-Shared Executive design.

### 3.2.3 Scheduling Considerations

The overriding concern in the scheduling philosophy of a Time-Shared system is equitable distribution of computer service to active terminals. This translates readily, from the users point-of-view, into reasonable response time. An important concept in the many approaches to the Time-Shared scheduling problem is time-slicing. Each user is allotted a small slice of time for use of the system. At the end of the time slice the program is suspended and system resources are given to the next eligible user. The program is continued when the Executive allots it another time-slice. This procedure is followed for all active users. The length of the time-slice or quantum is a critical decision and may in fact be dynamically adjusted by the Executive to suit the situation at hand. In a relatively unsophisticated form of round-robin scheduling, where the Executive attends to each terminal in a cyclic fashion, the slice of time may be solely a function of how many terminals are physically attached to the computer, and chosen to fix an upper limit on the average response time. It could be accomplished by dividing a set amount of time equally among the active terminals. Of course, this is an oversimplification, attempting only to illustrate a concept. Most Time-Shared scheduling algorithms are indeed more complex, taking into consideration the differences in modes of operation at the various terminals together with their accompanying response time requirements, computer resource balancing, and a priority structure.

### 3.2.4 Overhead

Overhead in terms of Executive code in a General-Purpose Time-Shared system is much the same as that of a General-Purpose Batch Processing system. A large amount of intricate code is characteristic. Modular design provides a solution to Executive memory requirements by allowing most of the Executive routines to be stored on high-speed bulk storage devices and swapped in and out of main memory in viable parcels as needed.

Overhead time can be somewhat disguised by the fact that no one terminal operator or his work is likely to utilize all facilities continuously. In an interactive environment, there is a considerable amount of "think" time expended on the part of the programmer at a terminal. During this "think" time, the Executive is free to perform functions unrelated to that particular user. Additionally, the printing of output to a terminal may be carried on under the auspices of the I/O channel with minimum Executive attention, providing further opportunity for Executive module swapping and performance of other support services classified as overhead. It is always necessary, of course, to fine tune those sections of Executive code which are frequently used or those which consume an unusual amount of time in relation to the service provided.

### 3.2.5 Storage Allocation

Given that Time-Shared System is obligated to provide service to a number of terminals simultaneously, an interesting problem presents itself. How is memory to be simultaneously available to all users? One approach is to divide it equally among the terminals thereby restricting each user to a fixed size partition which is a fraction of the real capacity of the equipment, and leave to the programmer the problem of fragmenting his work to fit the partition. However, since we are dealing with an illusion in a Time-Shared System, that of a total computer complex solely at one user's disposal, a more satisfactory approach is to extend that illusion to the apportionment of main memory by effectively giving each user all of main storage. The concept is known as virtual memory. Within the virtual memory context, the job is no longer limited in size to the amount of main memory available, but is bounded, in theory, only by the total of all high-speed storage, main and auxiliary, in the configuration. In practice, of course, the practical upper limit falls short of that mark. One implementation of the virtual memory concept is that of demand paging. Main memory is considered to consist of equal size blocks known as "page frames" which accommodate blocks of instruction code or data known as "pages". A job is automatically divided into pages. References made during program execution to instructions or data result in the movement of those pages containing the referenced material from auxiliary storage to core. Inactive pages may be moved out of core to make room for other applications. This can be done without any effort or knowledge on the part of the programmer. The Executive, possibly assisted by specialized hardware, performs the chores of ascertaining that a needed page is missing from memory, locating that page on auxiliary storage, moving it into available space in memory (perhaps, first making

space available by moving out a dormant page), and resolving the logical address reference with the actual physical location of the page. This technique permits all active terminal users to simultaneously operate on their programs or data, each viewing his job and data as entirely core resident; when in reality, memory is shared among all users with only certain active pages of each job present at any one time.

## 3.3 REAL-TIME SYSTEMS

A Real-Time System can be defined as one in which incoming data about an active environment is processed by the computer complex in sufficient time to provide results which may be used to immediately influence that environment.

That definition could be applied to a Time-Shared system, since a Time-Shared system is a form of Real-Time System. To further differentiate the type of Real-Time system to be considered, let us stipulate that it is not used in a "conversational" mode and that input and output may be directly from and/or to electrical or electromechanical equipment as well as from and/or to Human Beings. The primary function of such systems is control or assistance in control of a specific environment.

Real-Time data processing is a valuable tool wherever precise decisions must be instantaneously made concerning large, complex, and rapidly changing systems.

### 3.3.1 Response Time

It can be seen from our definition of real-time systems that the response time requirements are rather strict. Certain critical automated processes within a given application may require a reply from the computer within milliseconds. Other less critical operations may be tolerant of longer delays; but seldom is the response time expected to exceed a few seconds. Required response time in Real-Time systems may be said to be immediate, where immediate is quantified by the nature of the operation and may range in unit from, say, microseconds to seconds. Man-machine interaction is of a different nature in Real-Time systems than it is in Time-Shared systems. There is no dynamic alteration of programs or experimentation with data. In a Real-Time situation, the man-machine interaction, if any at all, usually consists of a man-made inquiry or request and a machine reply; or perhaps, merely the entry of new data by man into the machine or the display, in some form, of results or status from machine to man. In the most rigid Real-Time environments, output is directly used to control a live process with no intervention by man except in a monitoring capacity.

### 3.3.2 Workload Implications

A Real-Time system is a special-purpose system. The specific application, acceptable limitations, and performance specifications are well defined in advance of its fabrication. It is designed to perform a particular overall function. There is no need to complicate the Executive with general-purpose code to support all possible categories of computer user and their different data base and device requirements. The programs constituting the application are delineated in the planning stage and can therefore be adequately provided for in the design of the Executive. The heaviest burdens placed on Executive, Hardware, and Application tasks are that of speed and reliability.

The speed of the entire computer system is ultimately limited by the hardware, but poorly designed or coded application programs or Executive modules can seriously hamper performance. Therefore, strict attention must be paid to efficient algorithms and program coding conventions in all areas of the total system.

Reliability in both hardware and software is of the utmost importance in Real-Time Systems. Failure of either can be disastrous. As a result, particularly thorough debugging and exhaustive testing of program modules is critical. Hardware redundancy is essential. A cooperative system of hardware and software error detection and automatic recovery facilities and procedures is indispensable.

### 3.3.3 Scheduling Considerations

A rapid scheduling process is fundamental in a Real-Time system. The attainment of a speedy scheduling mechanism is aided, however, by the special-purpose nature of the system. Since all the tasks comprising the system are known, the hierarchical structure of various tasks is determinable. The necessary order of execution and/or the relative priorities of the programs can be established and, for all but unusual circumstances, fixed. The scheduling process is thereby simplified and need not contain the elaborate decision making machinations of general-purpose systems. If requests or inquiries are to be made of the system during its operation, provision must also be made to schedule the inquiry related programs without degrading performance or interfering with critical tasks. Accommodating random inquiries and their respective response time constraints adds to the intricacy and overhead of the scheduling process.

### 3.3.4 Overhead

It is difficult to overlap or hide overhead in a Real-time environment. However, Executive complexity and volume of code are potentially minimized

in Real-time special-purpose systems because one is dealing with a specific application and well-defined requirements. Reasonable restrictions can be placed on use of languages, record formats, data base characteristics, and device selection without fear of hindering an unknown computer user. If need be, time spent in swapping modules can be reduced by making the Executive and much-used application tasks entirely core-resident. Fine tuning of Executive code is also a potential source of reduced overhead time.

Overhead in a Real-Time system, however, cannot be defined in terms of Executive performance alone. The combined performance of all tasks in the network must be considered. The programs and programming techniques in a Real-Time system are uniquely controllable, since there are no remote unidentifiable programmers during system development. Good programming practices can be enforced and each program in the system can be optimized individually and as a part of the whole integrated network. A poorly performing task can be isolated for improvement and overall system bottlenecks can be identified and eased.

### 3.3.5 Storage Allocation

Allocation of storage can be either static or dynamic in a Real-Time Special-Purpose system. The choice depends on response time constraints imposed by the environment, the amount of money available for equipment (particularly storage), the potential speed of the integrated hardware, and the amount of system's programmer time and effort to be expended.

A completely static arrangement has all Executive and Applications program modules entirely and continuously present in main memory at pre-established locations. All data input and output areas are fixed in size and location. This approach has the advantage of no I/O overhead for module swapping and thus increased speed of operation and decreased response time. It also requires minimum effort on the part of the Executive since programs are loaded once and never moved, except in the event of hardware failure. This lessens the quantity of Executive code as well as its complexity. The most obvious disadvantages are (1) that extremely accurate upper limit estimates must be made regarding the volume of input and output data to be processed by each task during peak periods so that adequate main storage may be set aside, (2) during non-peak periods much of main memory is unused, (3) large quantities of expensive high-speed storage must be attached to the system, (4) Expanding the software capability of the system may require additional main memory and possibly other hardware modifications to accommodate the main memory addition.

Dynamic storage allocation has none of the cited disadvantages of static allocation, unless all programs are expected to experience peak loads simultaneously. New programs can usually be added to the system without hardware modification or purchase of additional memory. Memory is allocated on an as-needed basis for programs and/or data, thereby allowing for processing of background or support jobs in available memory during non-peak conditions. This method promotes greater flexibility and efficiency in the general use of storage. Also, errors in peak-load estimates are less serious and peak-load growth can be reconciled. Disadvantages include (1) increased overhead in terms of both time and code, (2) potential degradation of response time, (3) additional complexity in the Executive which requires a sophisticated design and programming effort, as well as more lead time for coding, debugging, and shakedown, (4) debugging is more difficult due to the intricacy and dynamic character of the system.

## 3.4 SUMMARY COMMENTS

It should be evident at this point that Executive Systems are tailored to their operating environments. They can differ greatly in overall purpose and, therefore, in design philosophy and implementation primitives. The anticipated workload characteristics, or lack thereof, are for the most part peculiar to an installation and must be taken into account in the individual tailoring process. Systems vary, among other things, in their response time requirements or definition of reasonable service, in their scheduling goals and overhead considerations, and in their approach to storage allocation.

It is worthy of note, here, that the manner in which Executive services and algorithms are implemented usually depends heavily on the idiosyncrasies and features of the central processor and related hardware with which it is to co-exist. It should also be pointed out that many systems do not fall readily into the pure categories of Batch Processing, Time Shared, or Real Time systems, as described, but are actually combinations of those philosophies.

It follows then, that measurement and evaluation of a particular computer system must be an equally tailored undertaking or the results will be misleading, or irrelevant. A thorough knowledge of the system objectives is therefore required; as is an understanding of the employed software concepts and of the hardware features and potential. Application of measurement tools also requires familiarity with system implementation details.

## 4.0 OPERATIONAL CONCEPTS

Although Executive System designs depend heavily on the hardware with which they are involved and on the primary type of service function thev are to perform, the following operational concepts may be found in most Executive Systems.

### 4.1 Multiprogramming and Multiprocessing

Multiprogramming describes the process of maintaining more than one program concurrently in a state of activity on a given hardware system. This may be accomplished by software interleaving of program execution on a single processor, simultaneous execution of programs on several distinct processors in the same configuration, or a combination of both.

In a multiprogramming environment, the active tasks in the system contend with each other for use of hardware and software resources, particularly central processor time. An Executive system exerts control over the environment and achieves organized activity by monitoring program and equipment status and switching resources between jobs to balance the ever-changing supply and demand.

When more than one central processor is used the system is said to be a multiprocessor system. Multiprocessing is a way to achieve a form of multiprogramming and to increase the systems workload capacity. However, multiprogramming does not necessarily require or imply multiprocessing. When the system has a multiprocessor configuration, there exists, in addition to the software contention for processor use, a potential hardware contention between processors for access to memory modules.

### 4.2 Interrupt Processing

An interrupt is a hardware initiated transfer of processor control to a fixed memory location, usually within the Executive area. Interrupts occur randomly to signal a particular type of significant event, such as program termination, expiration of a specified time interval, completion of an I/O transaction, or operator intervention.

Most large-scale computer systems are interrupt-driven; that is, the interrupt system is the vehicle for placing central control for coordination of activity in the hands of Executive routines. For instance, the transfer, by the Executive, of central processor use from one task to another is normally triggered by an interrupt. Since there are a number of possible conditions which could cause an interrupt, the Executive, upon receipt of control, first determines the reason for a particular class of interrupt

and then proceeds on the prescribed course of action. After the Executive has responded to the interrupt condition, it passes control to one of the eligible tasks in the system. The task receiving control of the processor after the interrupt may not be the one which was executing when the interrupt occurred. The relative eligibility of tasks for execution is often altered as a result of the interrupt.

A classic example of this procedure is one in which a high-priority task becomes dormant awaiting the completion of an I/O operation. A lower priority task is assigned use of the central processor while the high-priority task waits. When the interrupt occurs, signaling the completion of the high-priority task's awaited I/O transaction, the lower-priority task's activity is suspended and control is returned to the high-priority task. When any interrupt occurs, the Executive saves all pertinent information about the state of the program which was interrupted so that it can be subsequently continued without losing information critical to its progress.

The interrupt mechanism is also the computer's line of communication with the outside world. An operator desiring dialogue with the Executive regarding the status of the system presses a button or flips a switch which generates an interrupt. The interrupt is routed to the operator communications routine of the Executive and contact has been established. Data acquisition or display equipment ready for information exchange, interrupts a processor to initiate the data transfer or acknowledge its receipt. In each case the Executive intervenes and routes the interrupt to its proper processing routine.

### 4.3 Protection

Protection became an important problem with the advent of multiprogramming. It is normally handled with the aid of special hardware facilities. The nature and use of these hardware protection facilities varies with computer design, but the results are similar overall. Protection capability is supplied in two areas, control and access.

Control protection is effected by the provision of two machine operation modes, a "privileged" or "executive" mode and a "user" or "program" mode. In the executive mode all instructions of the machine can be executed including those which actually cause the assignation of the processor to a particular task and those which change the mode or status of machine operation. In the user or program mode, all instructions which might interfere with the proper operation of the executive or other program tasks are prohibited. Control protection is usually enforced by causing a switch to the privileged mode of operation accompanied by an immediate transfer to an executive routine, whenever an attempt is made while in

user mode to execute a privileged or otherwise unacceptable instruction. This allows the Executive to obtain processor control for analysis of the offense and determination of the proper course of action while keeping the remaining tasks in the system operational. In such systems, a user requiring a service which implies the legitimate use of a privileged instruction, communicates a "request for service" to the Executive through a special "executive call" instruction. The instruction transfers control and a coded service request to the Executive for disposition, and the request is carried out on behalf of the user by the Executive.

Memory access protection is provided through various schemes which prevent any user from illegally reading or writing over another user's core resident programs or data. This may be accomplished by assigning a unique protection code to all the blocks of memory assigned to a particular user and disallowing access to those blocks by any other user. Protection is achieved by comparing each user's protection code with the code assigned to the storage block that the user is attempting to access. Other schemes use special registers to contain the address bounds of each program in the system and compare addresses for validity as each access is attempted. Most protection methods provide some means for users to voluntarily share data or programs with other users. Sharable information may also be protected to various degrees, i.e., read only, write only, execute only, or be made accessible only to certain other users through special "access" codes. If an illegal access is attempted, an interrupt will occur causing the Executive to take control and perform whatever action is necessary to maintain the integrity of the system.

## 4.4 Failure Detection and Recovery

Abnormal conditions in a computer system may arise from many different sources such as operator errors, program coding errors, device failures, channel failures, parity errors, and power fluctuations. In most large-scale systems, some provision is made for hardware and software components to work hand-in-hand in detection of errors and reduction of their impact.

The amount of error checking and recovery capability incorporated in a given system varies largely with the environment for which it is intended and local overhead vs reliability considerations. For example, a system crash in a General-Purpose Batch Processing facility is costly and annoying, but the collapse of a Real-Time system could have catastrophic consequences. It would, therefore, be justifiable to spend a greater proportion of equipment money and development effort in real-time systems on failure detection and automatic recovery capability.

Redundant hardware is a costly but effective means of reducing the risk of failure due to hardware error. Duplicate processors, memory modules, channel paths, and auxiliary devices can be integrated into the system in such a way as to permit reconfiguration in the event of malfunction in a critical component. Executive software aids in the reconfiguration process and also restarts, refreshes, and/or relocates affected program modules and data bases.

In systems where failure is less disastrous than in real-time applications, hardware redundancy is unusual. The existing hardware modules may be equipped with retry capability. For example, an error condition detected by the hardware in accessing memory could cause an automatic hardware retry of that access before declaring an error condition. If the failure persists, it is reported to Executive software for disposition. If the error is not repeated on a hardware retry, its occurrence can still be recorded by the Executive for subsequent reporting and analysis,but the system may proceed as though it never occurred.

All error conditions need not be candidates for Executive processing. Certain classes of error, such as invalid data in an input record could be more reasonably handled by the application program responsible for the data. Each computer installation must independently judge the relative merits of various levels of error detection and recovery facility as they relate to their operating environments.

## 5.0 ARTS III SYSTEM OVERVIEW

The ARTS III system (in multi-IOP or multi-CPM mode) may be classified as a real time multiprocessor system. It is real time by virtue of the fact that the outputs of its programs are immediately used to assist in control of the air traffic situation in a terminal area; and multiprocessor by definition, since in a multi-IOP or CPM mode, more than one processing element in the configuration may be actively engaged in program execution. It follows that most of the programs operate with rigid response time requirements and that the existing man-machine interaction is not "conversational" but rather of an inquiry-response or data entry-display nature.

Scheduling in the ARTS III multiprocessor system consists of task selection by an available processing element from one of two tables of eligible tasks, the planned task table and the pop-up task table. A planned task is one which is executed at regular intervals governed by some cyclic process such as a radar beacon scan. The planned tasks in the system are ordered in groups known as cycles. Within each group or cycle the tasks are arranged in a network which identifies their interdependencies in the following manner. Associated with each task in a cycle is a list of predecessor tasks and a list of successor tasks. A planned task is eligible for execution if it has no predecessors or when all of its predecessor tasks have been completed. When a given task itself has been processed, its list of successor tasks is used to find other tasks in that cycle which may be now eligible for execution. Some tasks are restricted for execution to a particular processor, or processor type, and therefore will not be selected in the search of the planned task table by available, but otherwise inappropriate, processors. All tasks in a cycle must be completed before a new cycle is begun and all processors operate on tasks in the same cycle. The pop-up task table consists of tasks which are executed aperiodically at the request of planned tasks, or possibly as a result of an external inquiry or request. An entry for the task is placed in the table only when an explicit request for its execution has been made. Each of these tasks has a "time-for-execution" value associated with it in the pop-up table as a result of the request; and the smallest time value of all the tasks in the table is stored separately as an indication of the next time at which a pop-task is to be scheduled for execution. Available processors select tasks from this table as each of the task's "time-for-execution" arrives. The pop-up task table's smallest time value is examined before

the planned task table is searched, thus providing a mechanism for high-priority entrance to a task. This relatively low-overhead type of scheduling procedure is possible because implicit in the special-purpose nature of the system is the fact that all programs, along with their resource requirements, are determined in advance of system operation. In addition to expressing the actual dependency one task has on another's completion, the predecessor-successor network can be used to purposely space processing of tasks which may have conflicting resource requirements, such as access to the same memory module, and thereby reduce a form of system inefficiency.

An attempt to minimize overhead is reflected in the static storage allocation scheme. All operational programs and data bases in the ARTS III system are memory resident at all times and located at fixed storage locations. They remain at the original load-time addresses unless a system failure recovery procedure dictates their reallocation. Address assignment and linkage resolution are effected from relocatable program modules and tables during the system build process and may be modified by the operator at that time to accommodate last minute hardware reconfiguration. An effort will be made to optimize the initial storage assignment at each ARTS III site in order to minimize memory access conflicts.

Interrupt handling in the ARTS III multiprocessor system is not entirely centralized. The interrupt processing module of the Executive will process all but the Executive Service Request (ESR) interrupts and the application program I/O interrupts. There is a separate Executive module for handling requests for service communicated to the Executive by the application tasks. Requests may be made to initiate or terminate I/O operations, pass operator messages, provide normal program termination, schedule a pop-up task, modify cycle advance criteria, extend task addressing capability, clear overflow designators after arithmetic operations, alter channel interrupt ability or channel assignment for a specified peripheral, request I/O interrupt capture, and obtain information about IOP and channel assignment for a specific peripheral. The task making the Executive Service Request will pass to the Executive Service Request module those parameters necessary to select the desired service option and provide additional information peculiar to the requested service. Interrupts associated with task I/O transactions will be processed directly by the responsible task. The control information necessary to route I/O interrupts to each task will be transmitted to the Executive via an ESR or established during the system build process. It is expected that this procedure will reduce system overhead by eliminating Executive screening of task related I/O interrupts.

Both control and memory access protection are provided in the ARTS III multiprocessor system. Control protection is addressed through the provision of a privileged mode of operation for certain Executive instructions. If execution of these instructions is attempted while in non-Executive mode, an interrupt will occur causing control to be transferred to the Executive. Most of the privileged instructions are I/O related. Read and write memory access protection is provided through the use of 16 "memory lockout registers". These registers each contain the address bounds of one 16k memory module and an indicator for read lockout and/or write lockout for each 2k set of addresses within a particular module. When a task is initiated the memory lockout registers are loaded by the Executive with the proper read and write lockout values for that task. As each storage access is made by the task the significant memory lockout register is examined to determine validity of the access. If the access is invalid, an interrupt will occur and control will be transferred to the Executive. Similar protection is also provided during an I/O operation to prevent an IOP from writing input data into a protected area.

When two or more processors are simultaneously executing programs, the possibility arises that more than one of the programs may require access to the same data base. This becomes a problem, for example, when one program wants to modify (write into) the data base while another wants to read it. Such conditions can be avoided in the ARTS III multiprocessor system through proper use of special access control instructions. Invoking the access control instructions results in access to the data base by a particular processor for read operations as long as no other processor is modifying the data base. However, a processor cannot gain read access if another processor already has write access; nor can a processor gain write access if any other processor has either read or write access.

Even though various protection approaches are employed in a system, errors and failures do occur. In a real time system such as ARTS III, failure detection and automatic recovery are extremely important. In order to minimize the impact of system malfunction both hardware and software tools are employed. For example, during system operation the Executive software monitors the time a processor spends on a task by consulting a "time-away-from exec" table. The table contains an entry for each processor in the system. Before a processor begins a task, it updates its entry in the table with a maximum "time-away-from-exec" value. The table is periodically searched by other processors in the configuration and, if the value provided by a processor has been exceeded, an error condition is declared. This procedure deters the occurrence of an "endless loop" in a processor. Hardware detected malfunctions

such as power tolerance error, illegal function code execution, privileged operation error, protection violation, and memory parity error are handled with the aid of Executive routines. The ARTS III multiprocessor system will employ additional hardware for malfunction recovery in the form of a Reconfiguration and Fault Detection Unit (RFDU). The purpose of this unit is to serve as a central monitor point where information regarding detection of major hardware errors is routed and displayed, and where hardware reconfiguration is implemented. Two IOPs in the system will be able to obtain information about the status of critical system elements. One or the other of these two IOPs will be interrupted when the RFDU encounters an error condition. The offending element (IOP, CPM or memory module) will be isolated from the system by the combined efforts of the RFDU and Executive recovery software and replaced with a redundant hardware element if one exists. In the event redundant hardware is unavailable, the Executive recovery routines will work hand in hand with the operational tasks to gracefully reduce the system workload by eliminating non-critical tasks and reasonably degrading the performance of critical applications. These techniques should provide enhanced reliability to the total system.

# APPENDIX B

## A Survey of Available System Simulation Packages and General-Purpose Simulation Languages

11 June 1971

## ADDENDA

After the publication of this report, SIMSCRIPT was sold by Simulation Associates to CACI, Inc. A new version of the SIMSCRIPT compiler known as SIMSCRIPT II.5 has been announced by CACI. The salient features of the latest version is that it utilizes double precision floating point arithmetic. Pricing data has not yet been obtained from CACI.

# I. INTRODUCTION

Computer simulation has proved to be a valuable tool in studying the behavior of dynamic systems. The technique provides a method for monitoring the state of a system over time. Alternatives to simulation are mathematical analysis, experimentation with either the actual system or a prototype model, or seat-of-the-pants intuitive judgement. All of these have limitations. Mathematical analysis may be intractable because of the complexity of the relationships within the system. Experimentation can be costly, and it may be difficult to measure critical factors such as queue statistics. The seat-of-the-pants approach carries a high risk.

The problem of computer system design and evaluation has become more complex due to the development of time-sharing, multiprocessing, and real-time systems. The cost of the equipment prohibits experimentation. Selection of a new system based on intuitive reasoning is also ruled out by the magnitude of the investment. A reliable technique in forecasting the performance of proposed systems is simulation.

Simulation, in differing levels of detail, can be applied to the problems of design, development and evaluation of computer system. Use of simulation in the design phase gives the analyst a better understanding of the interactions of the proposed system components. Since a simulation is a dynamic model, status "snapshots" of the simulated system may be obtained at prescribed time intervals. Periodic status data along with cumulative statistics provide a comprehensive picture of the operation of the system under study.

A simulation model might be utilized to locate bottlenecks and evaluate proposed solutions during system development. An installed operational system can be studied, evaluated, and fine-tuned by systems personnel at the user facility via simulation of selected system cross-sections emphasizing suspected performance weaknesses. In addition, the process of designing a simulation model for the purpose of actual system performance evaluation assists in identification of those operational characteristics which must be measured in order to make evaluation by any technique possible and meaningful.

This survey was performed in order to establish the state-of-the-art of the techniques for system simulation which might be appropriate for computer system performance evaluation and adaptable to in-house equipment. A description of the state-of-the-art of the most widely used general purpose simulation languages and system simulation packages is contained in this appendix. Criteria for evaluating the languages and packages are suggested and a few guidelines for selection are given.

## II. CRITERIA FOR EVALUATION

Each simulation application has unique requirements with respect to the complexity of the problem and the skills of the analysts involved. Also, computer facilities and budget restrictions may vary. The choice of a particular simulation language or a system simulation package will be made by evaluating the alternatives in terms of a set of selected criteria. In this section, several criteria for use in evaluating the alternative approaches are presented. The decision maker would select the factors which are critical for his particular situation. It is doubtful that the entire set would apply to one application.

A notation of SSP means that the criteria applies only to system simulation packages; SL applies only to simulation languages.

A. Scope

SL (1) Capability to model discrete and/or continuous systems

(2) Basic items that are simulated and their properties

SSP (3) Ability to modify basic elements available in system library

SL (4) Facility for collecting performance statistics

- (a) Utilization of facilities
- (b) Average queue length
- (c) Idle time of system components
- (d) Total cost for simulated time span

(5) Output capability

- (a) Automatic at end of run
- (b) Optional (report generator)

(6) Input format

- (a) Coding forms

SSP (b) Special language to describe the system

(7) Ability to handle complex computations

(8) Ability to include subroutines or programs written in another language

(9) Diagnostic and debugging aids

B. Implementation

(1) Prerequisite programming experience

(2) Prerequisite modeling experience

(3) Man-days of training required

(4) User preparation time for each run

(5) Compilation and execution speeds

(a) Relationship to object machine
(b) Batch mode for models

(6) Hardware and software requirements

(a) Memory
(b) Compiler
(c) Peripheral devices
(d) Specific machine

(7) Market history

(a) Length of time
(b) Users

(8) Cost

(a) Rent, lease, buy, or non-proprietary
SSP (b) Per run charge
(c) One-time study done by vendor

C. Technical Support

(1) Training

(a) Cost
(b) Courses available and duration
(c) Organization providing user education

(2) Documentation

(a) Language manual
(b) Textbook and sample problem

(c) Operating instructions
(d) Input or coding forms

(3) Maintenance and system support after training and implementation

(a) System updates
(b) Newsletter
(c) Technical advice

(4) User Association

## III. SYSTEM SIMULATION PACKAGES

In this section five system simulation packages are described: SCERT, SAM, CASE, ECSS and CSS. The material is presented in terms of structure, use of the package, and implementation. A summary section is also included to recap the important features of each package.

The section on Computer System Simulator is brief because there is very little literature available about it; it is not marketed as the others are. It is included in this report to illustrate the efforts of an individual firm, IBM, in developing an internal simulation capability. More material is available on SCERT, SAM and CASE since they were developed primarily to be marketed.

### A. Computer Simulation System

Computer Simulation System (CSS) was developed by IBM in response to the need of IBM system designers for an analytical tool for predicting and evaluating new computer systems. CSS provides a language and structure to model a large variety of computer systems at differing levels of detail. [15]

The program is written in 360 assembler and requires 256K bytes of core to simulate a system. Larger models of S/360 (i.e., 65,67) are most appropriate for running CSS because it is processor bound.

CSS is a proprietary program designed primarily for use by IBM personnel. Customers may use CSS only by submitting their system specification data to IBM. The program cannot be modified to meet unique customer requirements.

Since CSS was written and designed for internal company use, it is not marketed and special arrangements must be made to use it.

### B. Extendable Computer System Simulator

The Extendable Computer System Simulator (ECSS) was developed by RAND Corporation in order "to improve the ease and the speed with which one can develop a model of a computer system". [p. 5, 13] In particular, the developers of ECSS wanted to develop a system simulation package which would (1) have a simple input format, (2) minimize programming and debugging, (3) provide capabilities for extending the language, and (4) allow for a flow-oriented or event-driven model.

#### STRUCTURE

An ECSS simulation is composed of three sections. First, the definition section specifies the clock units, storage units (i.e., bytes or bits) and any other dimensional quantities that are required. The system

description section defines the number and types of components in the system, their operating characteristics and the communication links between the units. (ECSS does not include a library of hardware and software characteristics so the programmer must provide this information.) In the system description section, the elements of the system are grouped into subclasses, classes, and pools. The third section of an ECSS job is the Job Load Description which describes the operation of the system.

USING ECSS

ECSS is coded in a high level English-like language. The system is written in SIMSCRIPT so the programmer may code portions of his system in SIMSCRIPT. The current version of ECSS requires that the programmer resort to some SIMSCRIPT II programming to reflect software features of the simulated system.

All of the distribution functions used in SIMSCRIPT are available with ECSS. Performance statistics are accumulated in the same way (see Section IV, C, for details on SIMSCRIPT). Likewise, debugging and diagnostics are similar to SIMSCRIPT.

IMPLEMENTATION

Since ECSS is written in SIMSCRIPT II, a copy of SIMSCRIPT must be available to run ECSS. A copy of ECSS, a non-proprietary program, may be obtained from Rand Corp. for a small fee to cover handling.

SUMMARY

An advantage of using ECSS is that the analyst can test hypothetical operating systems and hardware. However, since the program contains no library, a feasibility check is not performed for standard hardware/software and the analyst must check the specifications himself as well as provide operating characteristics.

ECSS is easy to program and use since the input format is in English-like sentences and phrases, but object-time execution efficiency is sacrificed.

## C. Computer-Aided System Evaluation (CASE)

Computer Learning and Systems Corporation developed and markets CASE, a proprietary systems analysis package. It can be used in the areas of feasibility studies, overall systems design, equipment selection, configuration management and product planning for computer hardware and software. Batch-processing, multiprogramming, multiprocessing time-sharing and real-time systems can all be analyzed.

## STRUCTURE

CASE is best described in terms of its major logical components:

1. Definition of the System - The workload of the simulated system is described in whatever level of detail is known. Files are defined, the run sequence is specified, and the type of system is indicated. Run characteristics may include the source language, input and output devices and internal processing activity. Of these items, many may be left unspecified and CASE will supply the missing information to allow optimum processing.

The initial hardware configuration may be specified in general terms or it may be specified in very specific terms. Experience has shown that the initial configuration should be as general as possible and that CASE can assist the analyst in determining a more specific configuration. The items that may be specified include CPU, memory size, number and type of channels, number and type of magnetic tape drives, and number and type of card readers, printers and card punches.

2. CASE Simulation - Two analyzers make up the major part of CASE. There are an independent processing analyzer (IPA) for single run and batch mode systems, and a concurrent processing--real-time analyzer (CPA/RTA). The IPA analyzes control input, the initial configuration and workload to make certain that the configuration is feasible. If the simulated system is batch mode, IPA then simulates the processing. CPA/RTA takes the results of IPA for non-batch systems plus the scheduling and real-time controls to determine a work schedule and then simulates the workload in multiprogramming mode.

3. CASE Library - Characteristics of currently available hardware and software are in the CASE library. Included in the library are operating characteristics as well as pricing data.

4. System Optimization - After reviewing the performance of the simulated system the analyst may adjust the workload design or the configuration. These adjustments will be made after careful analysis of the information provided by the CASE output reports.

5. Reports - Analysis reports are generated with each run to provide the information needed to make the configuration and system design modifications required to lead to the desired evaluation objective. On the final run management reports are prepared to completely document the final solution.

## USING CASE

Computer Learning and Systems provides CASE training on a variety of levels. A brief description of the courses follows:

1. CASE Executive Seminar - This course is for managers who require a broad overview of the use of simulation techniques, and the results that can be expected. It briefly discusses simulation in systems design, analysis of CASE reports, multiprogramming scheduling, etc. (one day)

2. Introductory Seminar - This seminar is of prime interest to CASE customers and prospects who require a first level working knowledge of the system. These are technically oriented sessions.(two days)

3. Basic CASE Use - This course will permit experienced customer analyst personnel to use the system. It deals primarily with input preparation and analysis of results, and includes a discussion of the CASE library, CASE algorithms, and systems design philosophy using simulation. (five days)

4. Advanced CASE Use - This course is a more "in depth" course designed for the CASE user. It expands on the basic course and includes instruction in system optimization and the use of CASE as a design tool. Also included is the handling of special hardware/software situations. (three days)

5. Real Time CASE - This course discusses the building of the real time model, collection of input data, and analysis of system bottlenecks, stressing techniques for using CASE as a tool for the proper design and configuration of real time systems. (three days)

Users of CASE have reported that it is relatively easy to use. There are three basic input forms for coding the files, workload, and configuration.

To date there have been approximately 40 CASE users. They include North Carolina National Bank, First National City Bank, U.S. Forest Service and the U.S. Air Force.

## IMPLEMENTATION

CASE is written in FORTRAN and requires 200K bytes of core to run. To date, it has been implemented on IBM S/360 model 50, GE 600, CDC 6000, and Univac 1108. Technical support is provided by Computer Learning and Systems Corporation.

## SUMMARY

According to the available literature, CASE appears to be technically sound for simulating computer systems. However, CASE has only been on the market for a year so no firm conclusions may be drawn with respect to maintaining CASE at the state-of-the-art.

## D. Simulation Analysis Machine (SAM)

Applied Data Research Inc., introduced SAM in the summer of 1970. It is the newest of the system simulation packages. The problem areas which SAM is designed to handle include system configuration, file distribution, operating system design and program system design.

### STRUCTURE

The SAM system consists of five distinct elements or parts:

1. Model Library - This library of predefined hardware/software systems contains operating characteristics and costs of currently available equipment. The library includes predefined models of processors, peripheral equipment and software.

2. Translator - The configuration to be simulated is described in the SAM language. This input is converted into internal numeric representations by the translator.

3. Model Generator - The numeric representations created by the translator are input to the SAM Generator which establishes the relationships of the various modules to produce the complete model of the system under study. In effect, this section acts as a linkage editor.

4. Interpreter - This portion of SAM "executes" the model produced by the Generator. As the model is executed, performance statistics are compiled.

5. Data Analyzer and Reporter - The data gathered by the SAM Interpreter is processed and a series of standard reports describing the performance of the system may be selectively produced. In addition, the user may generate specialized reports suited to his particular needs using the collected data. The standard reports include CPU Summary, Cost Effectiveness Report and Facility Usage Report.

A useful feature of SAM is that completely hypothetical hardware and software can be tested by specifying the characteristics in the SAM language.

USING SAM

As mentioned above, SAM has its own language for inputting system designs. A five-day course is required for an experienced programmer to learn the SAM language. SAM allows the user to include his own FORTRAN or Assembly language modules in the generated model since SAM is FORTRAN IV compatible.

At the present time, Goddard Space Flight Center is using SAM; there are four other industrial customers.

IMPLEMENTATION

SAM is available from ADR Inc., under a leasing arrangement for $2500 per month with a three-month minimum. Varied leasing plans using only portions of the model library are also available. Single study contracts may be arranged.

SAM is written in FORTRAN and requires 225K bytes of core to run. It has been implemented on IBM S/360 model 50. If the client does not have in-house computer facilities to run SAM, he can use the system as installed at ADR's Computer Center in an interactive mode through a remote terminal.

SUMMARY

Conceptually, SAM appears to have some features not available in the other currently available packages, in particular, the ability to test hypothetical hardware and software.

## E. System and Computer Evaluation and Review Technique (SCERT)

Comress Inc. developed and markets SCERT, the first proprietary systems simulation package. The package provides the capabilities for (1) selecting hardware configurations to meet anticipated work load requirements, (2) designing new systems, including determining how existing hardware can be used more effectively, and (3) evaluating the performance of systems and programs currently in operation.

STRUCTURE

SCERT is a series of software packages which has been designed to simulate computer systems. As an integrated group of programs, it runs in many

different phases. Conceptually, however, SCERT can be viewed as consisting of five primary functional phases:

(1) Introduction of Processing Requirements - This first phase accepts input definitions outlining the workloads and processing requirements of the system to be simulated. Characteristics of the files as well as the systems environment (i.e., programmer experience and salaries), frequency of each job, and internal processing activity are specified in this phase. A mathematical model of each computer run is built and validated.

(2) Introduction of Hardware/Software To Be Simulated - In this phase the hardware and software configuration is introduced. Software packages such as sort routines, the operating system, and the compilers to be used in program preparation are specified. In addition, channel assignments and the communications network can be described. The SCERT factor library contains performance specifications for all currently available hardware and software. The SCERT simulation uses data from this factor library to build a mathematical model of the hardware/software configuration and to validate the compatibility of the models built in PHASE I with the configuration.

(3) Presimulation Algorithms - The models built in PHASE I are passed against the model of the hardware and software. A series of calculations which structure and parameter the nonhardware-oriented models to the performance abilities are performed.

(4) Simulation - The data and models of the prior phases are incorporated to simulate processing. Whenever multiprogramming or multiprocessing are simulated, a special SCERT routine is employed to reflect these features.

(5) Output Reports - SCERT has been designed to produce a series of standard reports. The reports available for a particular run depend upon the type of system being simulated (batch, multiprogramming, multiprocessing, time-sharing) and the options selected by the analyst. The reports have been designed to provide information which is useful in evaluating the system efficiency. For example, the Computer Capabilities Report tabulates simulated break-out of thru-put requirements. This report provides a means to pinpoint critical hardware areas. Table 1 summarizes the standard SCERT output options.

If the analyst wants to evaluate the characteristics of a hypothetical configuration, there are several alternatives available. The easiest would be to modify the system software represented in the simulation. Another possibility would be to have Comress analysts enter the proposed elements in the factor library. Finally, the results of an external simulation program written in another language could be incorporated into the SCERT input.

Table 1
SUMMARY OF STANDARD OUTPUTS

| Report Title | Level of Detail | Orientation | | |
|---|---|---|---|---|
| | | Primary | Secondary | Minor |
| Computer Complement | Intermediate | Audit | Cost/Performance | Documentation |
| Central Processor Utilization | Intermediate | Cost/Performance | Analysis | |
| Programming Requirements | Intermediate | Implementation | Cost/Performance | |
| Application Summary | High | Implementation | Cost/Performance | |
| Computer Capabilities | Intermediate | Analysis | Cost/Performance | |
| Cost Summary | High | Cost/Performance | | |
| Real Time Analysis | | | | |
| Event Processing | Intermediate | Analysis | Audit | |
| Hardware Utilization | Intermediate | Analysis | Cost/Performance | |
| Systems Response | Intermediate | Cost/Performance | Analysis | |
| Memory Requirements | Intermediate | Analysis | Implementation | |
| Multi-Programming Schedule | Intermediate | Audit | Implementation | Analysis |
| Detailed Analysis | Lowest | Audit | Analysis | Implementation |
| Summary Analysis | High | Cost/Performance | | |
| System Documentation | Lowest | Documentation | | |

## USING SCERT

SCERT has its own language for inputting system specifications. A seasoned programmer analyst with about three years of experience can learn to use and apply SCERT after completion of a two-week training course offered by Comress, the vendor of SCERT. Comress provides documentation and input forms as well as customer education.

If the user intends to program SCERT (rather than hiring Comress analysts to perform the study), customer education classes are available at $500 per person for a ten-day introductory course. The first week of the introductory course includes the SCERT language and input forms. In the second week analyst training for modelling batch and multiprogramming systems is given. An additional one-week post-graduate course is offered for modeling time-sharing systems.

At the present time, there are approximately 80 users of SCERT; to date there have been 400 SCERT clients. In the Boston area companies that have used SCERT include Hartford Insurance Co., John Hancock Insurance Co., State Street Bank, and New England Telephone Co.

## IMPLEMENTATION

SCERT is marketed by Comress. It can be leased or a study contract can be arranged with Comress analysts. There is a one-year minimum for leasing SCERT, and the rental ranges from $1500 to $3000 per month depending upon what portion of the factor library is used. Table 2 summarizes the different leasing plans.

SCERT requires 110K bytes of memory and can be run on IBM S/360 model 40 and up under OS or DOS. In addition, it has been implemented on the UNIVAC 1108 and RCA Spectra 70/45. If on-site facilities are not available for SCERT, customers may submit their runs to Comress who in turn has them run at a local service bureau. If this procedure is used, there is a $500 reduction on the monthly lease.

Comress provides monthly updates to the factor library. Analyst support is available at $250 per day. A large technical staff is also available to aid clients with implementation problems. In addition, there is a SCERT users' association.

## SUMMARY

SCERT is the oldest proprietary simulation package available. The experience of users has been incorporated into the design of the system, and the result is a highly sophisticated system backed by an experienced technical staff. There are several features of this system that should be considered when selecting a system simulation technique.

(1) If model does not contain a parameter to reflect a particular function, or to adjust some activity the user can do little to overcome the difficulty. However, Comress is sensitive to clients' needs and might be willing to modify the SCERT program to handle the situation.

(2) Comress has a very capable technical support staff.

(3) An in-house computer is not required to use SCERT.

Table 2
SCERT LEASING PLANS

| Type | Monthly Rental Fee[1] | Factor Library | Education Included[2] | Sets of Documentation |
|---|---|---|---|---|
| Basic | $1,500 | Any 2 general CPU | None | 1 |
| Flexible | $2,000 | Any 2 CPU or all general CPU | 1 | 1 |
| Comprehensive | $3,000[3] | Entire factor library | 2 | 1 |

[1] Based on 1-year lease. Deduct $500 if not run in-house.
[2] Number of 2-week sessions. Multiple copies of SCERT within an organization reduces the rental charge.
[3] For TSC this fee would be $1,000 since DOT already leases SCERT.

## IV. GENERAL-PURPOSE SIMULATION LANGUAGES

The descriptive material in this section is organized like that in the previous one. Four simulation languages are discussed: GPSS, SIMSCRIPT, GASP, and HOCUS. HOCUS was introduced in the United States in the fall of 1970 so there are not yet any HOCUS customers and little material is available.

### A. Hand or Computer Universal Simulator (HOCUS)

HOCUS was developed by P-E Consulting Group Ltd. and has been used by a number of European firms. It is particularly useful for simulating process models. Since no programming experience is required, HOCUS can be used and understood by line management. Models can be rapidly set up and run by less specialized analysts using a simple logic language.

HOCUS uses the concepts of entities and activities (like SIMSCRIPT and GASP). However, HOCUS provides only a limited simulation capability with far less sophistication than GPSS or SIMSCRIPT. It requires only 8K words of storage and a FORTRAN compiler so it can be run on a small computer. Some HOCUS clients use HOCUS for initial simulation efforts and then convert their models to GPSS or SIMSCRIPT.

The purchase price for HOCUS is $14,000 which includes 5 days of consulting. It may also be leased on a quarterly basis for $2000 per quarter including 2 days of consulting. A three-day HOCUS training course is available for $290 per student or on-site for $2500.

### B. GASP

GASP was developed at U.S. Steel Corporation in 1963 in response to a need for a language to simulate steel manufacturing processes. At that time, there was no general-purpose simulation language available so the developers of GASP merely expanded upon an existing scientific language, FORTRAN.

#### STRUCTURE

GASP formalizes an approach to simulation by specifying common elements of simulation studies and providing subprogram tasks that are independent of individual problems; it is not a language. Twenty-three FORTRAN subprograms linked and organized by a main program comprise the system.

GASP depicts the world as made up of entities that are described by attributes and related through files. The status of the simulated system can be changed if entities are created or destroyed, if attribute values change or if file contents are altered.

Elements, the resources of the simulated system, may be temporary or permanent. Some examples of elements are the CPU, program module, and card reader in a computer system. Each of these items has distinguishing characteristics or attributes. The CPU has a fixed number of registers; the card reader has a maximum speed, and a program module has a core requirement. Events, such as a program utilizing the CPU, cause the status of the system to change.

GASP provides a simulation timer. The programmer may control the length of a run through this timer which represents "simulation time" or by the number of events.

At the termination of a run, standard output is automatically produced. It includes the mean, variance, maximum and minimum of simulation-generated data as well as a frequency count for this data. Contents of all queues and the maximum and average queue length are also provided. Additional output may be programmed in FORTRAN.

The size of a GASP model depends upon the core capacity of the object machine. GASP is usually used on small computers which cannot support a more complex simulation language.

## USING GASP

A working knowledge of FORTRAN is a prerequisite to using GASP. Since GASP is composed of FORTRAN subroutines, an inexperienced programmer could not use the language. However, an experienced FORTRAN programmer will have no difficulty in learning to use GASP.

GASP utilizes the FORTRAN compiler so diagnostics are limited to those provided by the compiler. Additional GASP features to facilitate debugging and error detection are (1) automatic monitoring of program variables, (2) selective tracing of program flow, and (3) programmed dumping of all system variables. The same time consuming compilation runs and lengthy debugging experienced with FORTRAN occur with GASP.

Simulation Associates offers a two-day seminar on GASP. (Cost of the session is $175.) The seminar is designed primarily for programmers, engineers and systems analysts and provides a working knowledge of GASP. An industrious FORTRAN programmer could probably learn to use GASP by reading the language manual and an available text (see ref. 14).

## IMPLEMENTATION

GASP is a non-proprietary program. The program may also be obtained from Simulation Associates for $25.

The language is particularly suitable for use on small computers for which no other simulation language exists. It has been implemented on the IBM 1130, 1800 and S/360, GS 225 and 415; SDS 930, and CDC 3400. The

only requirements for implementation are a FORTRAN compiler and 8K words of memory.

Since GASP is merely a collection of FORTRAN subroutines, the subroutines could be used for any purpose.

SUMMARY

There are several features of GASP which should be kept in mind when selecting a simulation language:

1. A knowledge of FORTRAN is required to use GASP.

2. GASP can be implemented on any computer with a FORTRAN compiler.

3. Debugging is time-consuming.

4. GASP is not suitable for complex large-scale simulations. Higher level simulation languages provide more modeling statements and are able to simulate large systems more efficiently than GASP.

The structure of SIMSCRIPT is very much like GASP. Installations with small computers may want to test skeleton models of large systems using GASP in-house and then convert them to SIMSCRIPT for the full-blown model on a larger system.

5. GASP is best suited for organizations with infrequent simulation study requirements and organizations with limited computer facilities.

C. SIMSCRIPT

SIMSCRIPT was developed by the RAND Corporation to meet the demand for a general-purpose simulation language. It is rated by some as the most powerful simulation language now generally available.

STRUCTURE

SIMSCRIPT is an event-oriented simulation language for analyzing discrete systems. (An event-oriented language monitors the status of a system and the effect of events on the system.) The language can be broken down into the following basic elements:

1. Entities - The resources of the simulated system. Entities may be equipment or items that use the equipment and may be temporary or permanent. Temporary entities are created and destroyed during the execution of a simulation while permanent entities remain during the run. Each entity is described by attributes which may be temporary or permanent. A card reader, tape drive and program are examples of entities in a computer system.

2. Sets - Groups of Entities. All of the tape drives in a computer system would be a set.

3. Event Routines - Activities of the system. Processing of jobs would be represented by a series of activities. Events may be endogenous, arising from actions within the system, or exogenous, arising from actions in the system environment. An endogenous event would be a program using a peripheral device while an exogenous event would be an operator interrupt.

The sequence of events in real-time is represented by creating and destroying temporary entities. For example, in a simulated multiprocessor, a program module would be a temporary entity. It would be "created" when it was ready for input processing and "destroyed" when output was completed. Simulation time is monitored by the simulator clock which assumes time is expressed in terms of days, hours, and minutes. The clock units may be redefined according to the user's needs.

There is no standard output at the end of a simulation run. All output reports are produced by the report generator and must be formatted by the programmer. The report generator provides the capability to produce sophisticated output, but an equivalent amount of sophisticated coding is also required.

All of the computational facilities of FORTRAN that might be required for simulation have been included in the SIMSCRIPT language. If necessary, FORTRAN subroutines can be incorporated into the model. A time saving feature of SIMSCRIPT is that portions of a model may be compiled and retained for future runs rather than recompiling with every run.

## USING SIMSCRIPT

SIMSCRIPT is designed for an experienced FORTRAN programmer. The language is complex and an inexperienced programmer would have difficulty in learning to use it. Even experienced programmers have found debugging a SIMSCRIPT model to be difficult since diagnostics are limited. SIMSCRIPT II Plus, available from Simulation Associates, includes extensive compilation and execution diagnostics designed to reduce designing time. The lack of structure of the language also adds to the debugging time required.

There are quite a few good SIMSCRIPT language manuals available. A SIMSCRIPT language manual, application description, and operator's manual may be obtained from the RAND Corporation. In addition, SIMSCRIPT textbooks have been published by Prentice-Hall (see Kiviat, Villaneuva & Markowitz, The SIMSCRIPT II Programming Language and Shukiar, A FORTRAN Programmer's Introduction to SIMSCRIPT II).

Simulation Associates distributes this book to all SIMSCRIPT II Plus clients. The text is broken down into five separate levels with each level keyed to the degree of experience in simulation programming.

Two courses on SIMSCRIPT II are available from Simulation Associates. An introductory 2-day seminar is designed to provide an overview of the language. A 5-day in-depth course is also offered for programmers, systems analysts, and O.R. analysts involved in the study of complex systems. The costs for the introductory and in-depth courses are $200 and $425 respectively.

IMPLEMENTATION

The public domain version of SIMSCRIPT II is available from SHARE. SHARE merely distributed copies of the language and provides no maintenance or support. This version of SIMSCRIPT requires 150K bytes of memory. It has been implemented on (1) CDC 3600, 6400, 6600, 6800; (2) UNIVAC 1107, 1108; (3) GE 625, 635, and (4) IBM S/360.

SIMSCRIPT II Plus has been marketed for about one year. Approximately 40 companies are using SIMSCRIPT II Plus on a trial basis, and eight have decided to lease it.

SIMSCRIPT II Plus, including complete technical support, is available from Simulation Associates. The cost for the service is $6,000 for the first year; this fee declines by $500 per year until $25,000 has been paid by the end of the fifth year. The price beyond the fifth year is $1,000 per year. The same services can be purchased on a monthly basis at $600 per month.

If suitable in-house facilities are not available, SIMSCRIPT II Plus can be used through the time-sharing facilities of Computer Software Systems, Inc. CSS leases SIMSCRIPT from Simulation Associates. Therefore, the only charges incurred in using SIMSCRIPT through CSS are the normal time-sharing rates. By using CSS the user also has the advantage of debugging his model in a time-sharing mode. CSS rates are the following:

time-sharing mode:
$.38 per CPU second
$15 per connect hour
batch mode:
$.24 per CPU second if run between 6-8 p.m.
$.16 per CPU second if run overnight
plus I/O charges for cards and printer.

At present, SIMSCRIPT II Plus is available only for IBM System/360 computers with at least 256K bytes of core. A version for RCA equipment will be available in January 1971. Simulation Associates can rewrite this language for other computers under an implementation contract.

The major differences between SIMSCRIPT II Plus and the public domain version of SIMSCRIPT are the following:

1. Simulation Associates reports that compared with the public domain version, SIMSCRIPT II Plus requires 85% of the execution time, 40% of the compilation time, and only 5% of the assembly time. These performance improvements are significant enough to justify purchase of SIMSCRIPT II Plus for any organization that will be making substantial use of the program, since the savings in computer time can more than offset the cost of leasing the program.

2. SIMSCRIPT II Plus will operate under any version of OS/360; the public domain version will not operate properly under MVT or MFT II.

3. Simulation Associates provides full installation and maintenance support for SIMSCRIPT II Plus. The public domain version is totally unsupported.

4. Additional instructions have been added to SIMSCRIPT II plus, along with powerful debugging features such as a traceback to the source statement in which an error occurred.

SUMMARY

There are several salient features of SIMSCRIPT which should be kept in mind when considering the use of this language. They include the following:

1. An inexperienced programmer could not easily learn SIMSCRIPT.

2. All output must be specified by the user.

3. Debugging is time consuming.

4. There are a number of good language manuals available for SIMSCRIPT.

5. SIMSCRIPT has been implemented on most large machines.

6. SIMSCRIPT is a powerful simulation language designed to model large-scale systems.

7. A large model computer is required to run SIMSCRIPT.

8. SIMSCRIPT II Plus is obviously superior to SIMSCRIPT II. The extent to which the language will be used should be considered when deciding whether or not the advantages of the pay version justify the fee.

## D. General-Purpose Simulation System

One of the oldest and most widely used simulation languages is IBM's General-Purpose Simulation System (GPSS). GPSS is a language designed specifically for simulating discrete systems. The applications of GPSS are limited by the ingenuity of the programmer rather than by the scope of the language.

### STRUCTURE

GPSS is a transaction-oriented language; a transaction-oriented language is designed around entities moving through the system. A transaction might be a message, a customer, a cargo ship, parts moving through a factory, or a program module. In contrast an event-oriented language monitors the status of a system as events effect the system.

The language can be broken down into the following elements:

1. transactions - the units of traffic that are created and moved through blocks by the GPSS program (i.e., a message)

2. facilities - represent equipment which can be used by only one transaction at a time (i.e., a card reader, a bank teller, a stock clerk)

3. storages - represent equipment that can service more than one transaction at a time (i.e., core storage in CPU)

4. random number generators - 8 "pseudo-random" number generators which generate repeatable series of numbers

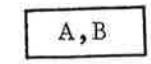
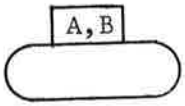
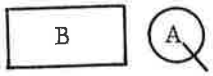
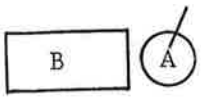
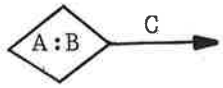
5. blocks - 43 specific block types each of which represents a characteristic activity of a system. The block types include entering and leaving queues, using and leaving a facility or storage, transferring to another part of the system and tabulating results (see Table 3). Each block type has a unique block-diagram symbol so that GPSS models can be flow-charted before coding is begun.

6. standard numerical attributes - are used to save and report information such as the total number of transactions that entered and left a queue

7. chains - groups of transactions in the simulated system

Table 3

SOME GPSS BLOCK-DIAGRAM SYMBOLS

| BLOCK TYPE | SYMBOL |
|---|---|
| ADVANCE |   |
| ASSIGN |   |
| DEPART |   |
| LEAVE |   |
| QUEUE |   |
| TEST |   |

Entities or transactions move through the simulated system. The sequence of events in real-time is represented by the movement of transactions from block to block in simulated time. For example, in a simulated multiprocessor computer system, a program module would be the transaction. The path of the program in the system might entail waiting for the CPU, accessing peripheral devices, and calling additional core resident modules. The programmer can select any time unit he desires. Time might be in seconds, minutes or nano-seconds.

GPSS automatically collects and prints a standard statistical output unless the user chooses to suppress it. The standard output includes the following:

1. average utilization, total number of entries and average usage time per transaction for each facility.

2. average contents, average utilization, total number of entries, maximum contents, and average usage time per transaction for each storage.

3. for each queue, maximum contents, average contents, total entries, average time in queue per transaction, and contents of the queue at the termination of the model.

4. mean, standard deviation, number of entries and frequency tabulation for each table.

5. the value of any savevalues (savevalues are variables set aside by the user to save specific values).

Using the report generator the programmer can receive graphical output as well as select and title which of the standard outputs he wishes to receive. For a simple model, the standard output is usually sufficient.

The size of a GPSS model depends upon the core capacity of the machine used.

USING GPSS

GPSS is specifically designed for a new programmer and no prior computer experience is required. The unique flowcharting scheme greatly facilitates coding. Standardized coding forms are available from IBM.

Virtually any system may be modeled using GPSS. However, if the programmer finds that a particular computation is cumbersome to perform in GPSS, he can incorporate 360 Assembler or FORTRAN subroutines into the GPSS model.

IBM offers a 5-day GPSS training course at a cost of $340 per student. A similar course is also offered by Simulation Associates. Due to the lack of instructional material, such a course is mandatory for any programmer attempting to use GPSS. The only documentation available from IBM are the GPSS Users Manual and System Manual; both are designed as reference manuals not instruction texts. No programmed instruction course is currently available. An introductory textbook [16] has been published in a preliminary edition and is a tremendous improvement over available IBM materials. A 2-day advanced course on GPSS is also offered by IBM.

Coding and debugging a model require a relatively short time compared to a scientific language such as FORTRAN or ALGOL. Debugging a GPSS model is facilitated by the excellent diagnostic messages. If desired, models may be batch run.

The time required to assemble and run a GPSS model depends upon: (1) the machine being used, (2) the number of blocks in the model, (3) the complexity of the conditional logic in the model, and (4) the number of queues in the model. No meaningful time estimates can be made since all GPSS models vary in length and complexity.

## IMPLEMENTATION

GPSS is marketed by IBM. The most recent version of GPSS (GPSS/360 version 2) is available for S/360 and is rented by IBM for $20 per month. The source language is 360 Assembler. GPSS runs on S/360 (DOS or OS) with a minimum of 64K bytes of memory. GPSS/360 version 1 is available free to S/360 customers, but it is no longer maintained by IBM. GPSS II, an earlier version of GPSS written in FORTRAN, has been implemented for the Univac 1107 and 1108. It is available from Univac on a rental basis.

GPSS/360 version 2 provides a few advantages over version 1. The significant differences are the following:

1. A real-time TIMER for model interruption and continuation.

2. Ability to incorporate independently compiled FORTRAN subroutines.

3. Improved diagnostic aids.

4. Provision for GPSS/360 data sets on 2314 direct access storage devices. (Version 1 assumes 2311 storage devices but this can be altered at run time.)

The improvements in version 2 are not critical to the user.

## SUMMARY

Before selecting to use GPSS for a particular problem, the analyst should be aware of the following advantages and disadvantages of using the language:

| | SYSTEM SIMULATION PACKAGES | | | | LANGUAGES | |
|---|---|---|---|---|---|---|
| | CASE | SCERT | SAM | ECSS | GPSS | SIMSCRIPT | GASP |
| Objects being Simulated | computer hardware/software, i.e., C.P.U., peripherals, operating system | | can test hypothetical elements | | transactions<br>facilities<br>storages<br>user groups<br>queues | events<br>entities<br>sets | events<br>elements<br>files |
| Properties of Objects | cost, speed, interface compatibility<br>CASE library | SCERT factor library | SAM library | defined by user | parameters & priority of transactions, capacity of storage | attributes | attributes |
| Report Output | Analysis reports & mgt. reports | Select std. reports or program specially tailored ones | standard & specialized reports | same as SIM-SCRIPT | Queue statistics<br>Facility "<br>Storage "<br>Frequency "<br>Clocktime<br>Report generator | user must program | GASP summary, FORTRAN output statements |
| Input format | fill-in-the-form for files, workload & configuration | fill-in-the-form | SAM-language | English-like language | GPSS code | SIMSCRIPT code | FORTRAN statements |
| Compatibility with other languages | none | none | FORTRAN | SIMSCRIPT | Assembler, FORTRAN (pay version only) | FORTRAN, Assembler | FORTRAN |
| Diagnostic & debugging aids | hardware/software compatibility is checked | hardware/software compatibility is checked | produced by translator to check logic & oper. of system | same as SIM-SCRIPT | very explicit diagnostics<br>trace feature | fair diagnostic debugging can be time-consuming unless II Plus version is used | FORTRAN diagnostics, System snapshots |
| Programming Requirements | Programming experience | 3 years computer experience | programming experience | SIMSCRIPT, FORTRAN | none | FORTRAN | FORTRAN |
| Training Required | 5-day course | 10-day course; optional 1-week course | 1-week course | same as SIM-SCRIPT | 1-week course | 1-week course | knowledge of FORTRAN |

| | SYSTEM SIMULATION PACKAGES | | | | LANGUAGES | |
|---|---|---|---|---|---|---|
| | CASE | SCERT | SAM | ECSS | GPSS | SIMSCRIPT | GASP |
| Manuals & Textbooks | available when package is leased | | | | IBM doc. text available | several avail. texts, manuals from SIMULATION Assoc. & Rand Corp. | text |
| Cost | Lease | Lease | lease | $100 from Rand for tape | free to S/360 users, also pay version @ $20/ mo. | free to SHARE, pay version from Sim. Assoc. | $25 - one-time fee |
| Originating Organization | Comp. Learning & Systems Corp. | Comress | ADR | Rand | IBM | SHARE, Sim. Assoc. | Sim. Assoc. |
| Machines Implemented | S/360-50, GE 600, CDC 6000 UNIVAC 1108 | S/360-40 110K bytes core UNIVAC 1108 RCA Spectra 70/45 | S/360-50, 225K bytes Core + FTN | Same as SIM-SCRIPT | S/360 model 30 + up (64K bytes memory) | 150K bytes S/360 CDC + UNIVAC | Any with FORTRAN Compiler & 8K words memory |

## VI. Some Guidelines for Selection of A Simulation Capability

Every installation has unique simulation requirements. Depending upon the related criteria and simulation applications, the appropriate plan might be the use of a system simulation package as well as a language, or only one of these approaches. In selecting the appropriate capability for a particular installation we suggest the following guidelines:

1. If there is a need for extensive complex system simulation, a package is probably the best method. There is no development time lag or risk with a tested package.

2. The currently available packages, CASE, SAM, and SCERT, appear to be about the same technically and pricewise, but SCERT has been proven worthwhile on the market while the other two are new in the field.

3. Users of the packages and languages prove to be an excellent source of information about the utilization of the techniques and should be contacted.

4. HOCUS and GASP are useful for small computers (8K words). HOCUS is extremely easy for non-computer people to use, but it is expensive. If experienced FORTRAN programmers are available, problems can be analyzed using GASP.

5. SIMSCRIPT and GPSS are the most powerful simulation languages and, if feasible, one should be selected. The selection of one over the other depends upon: (a) the in-house computer available, (b) the experience of the analysts involved, (c) the complexity of the model to be analyzed, and (d) the budget allocation.

If in-house computer facilities will not support GPSS or SIMSCRIPT, prototype models might be tested using GASP in-house. Then the full-blown model could be written in one of the sophisticated languages and run on an outside computer. Budget limitations would determine whether or not SIMSCRIPT II Plus could be rented and how many employees could participate in language classes.

6. Evaluate capabilities of the available computer facility to determine which of the alternatives could be run in-house.

7. Evaluate in-house programming skills to determine the related educational requirements for each alternative.

## Bibliography On Simulation


1. GPSS/360 User's Manual, IBM Form Number H20-0326.

2. SCERT: Systems and Computers Evaluation and Review Technique, Comress, Inc., Washington, D.C., 1967.

3. Bairstow, Jeffrey N., "A review of systems evaluation packages," Computer Decisions, June 1970, p. 20.

4. Boehm, B.W., Computer Systems Analysis Methodology: Studies in Measuring, Evaluating, and Simulating Computer Systems, R-520-NASA, Rand Corp., Santa Monica, 1970.

5. Gordon, Geoffrey, System Simulation, Prentice-Hall, Englewood Cliffs, 1969.

6. Gould, R.L., "GPSS/360 - an improved general-purpose simulator," IBM Systems Journal, 8, 1 (1969), pp. 16-27.

7. Hutchinson, George K., & Maguire, John Norris, "Computer Systems Design and Analysis Through Simulation," Proceedings of Fall Joint Computer Conference, 1965, pp. 161-167.

8. Kiviat, Philip J., Villaneuva, R., & Markowitz, Harry M., The Simscript II Programming Language, Prentice-Hall, Englewood Cliffs, 1969.

9. MacDougall, M.H., "Computer System Simulation: An Introduction," Computing Surveys, 2,3 (Sept. 1970), pp. 211-242.

10. Markowitz, Harry M., Hausner, Bernard, & Karr, Herbert W., SIMSCRIPT - A Simulation Programming Language, Prentice-Hall, Englewood Cliffs, 1963.

11. Merikallio, Reino A., "Simulation Design of A Multiprocessing System," Fall Joint Computer Conference, 1968, pp. 1399-1410.

12. Naylor, J.H. et al, Computer Simulation Techniques, John Wiley and Sons, New York, 1966.

13. Nielson, N.R., ECSS: An Extendable Computer System Simulator, RM-6132-NASA, Rand Corp., Santa Monica, 1970.

14. Pritsker, A. Alan B. & Kiviat, Philip J., Simulation With GASP II, Prentice-Hall, Englewood Cliffs, 1969.

15. Seaman, P.H., Sourcy, R.C., "Simulating Operating Systems," IBM Systems Journal, 8,4 (1969), pp. 264-279.

16. Schriber, Thomas J., GPSS/360 Introductory Concepts and Case Studies, University of Michigan, Ann Arbor, 1968 (Preliminary Edition).

17. Shukiar, H.J., A FORTRAN Programmer's Introduction to SIMSCRIPT II, RM-5937-PR, Rand Corp., Santa Monica, 1969.

18. Teichroew, Daniel & Lubin, John Francis, "Computer Simulation - discussion of the technique and comparison of languages," Communications of the Association for Computing Machinery, 9, 10 (Oct. 1966), pp. 723-741.