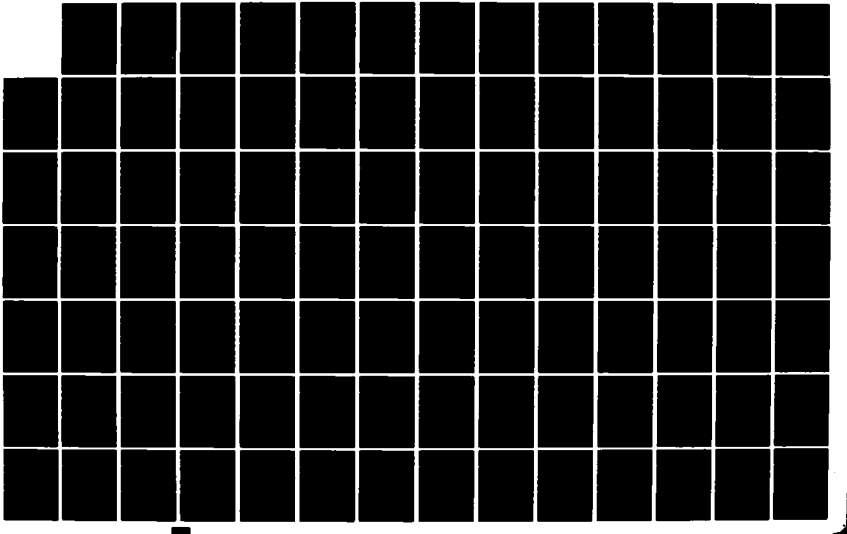
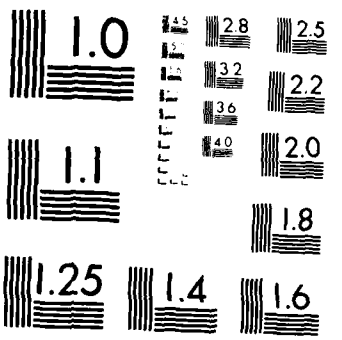
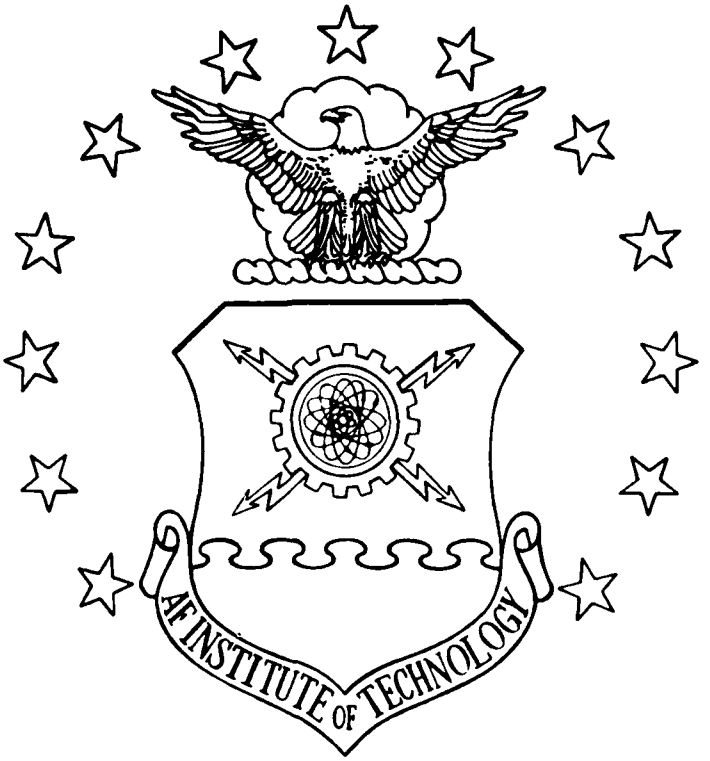
AD-A151 899 DEVELOPMENT OF A USER SUPPORT PACKAGE FOR CPESIM II (A 1/2
COMPUTER SIMULATIO.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. D L PETTY
UNCLASSIFIED DEC 84 AFIT/GCS/ENG/84D-21 F/G 9/2 NL







DEVELOPMENT OF A USER SUPPORT PACKAGE FOR CPESIM II (A COMPUTER SIMULATION FOR CPE USE)

THESIS

Daniel L. Petty
Captain, USAF

AFIT/GCS/ENG/84D-21

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

DEVELOPMENT OF A USER SUPPORT PACKAGE FOR CPESIM II (A COMPUTER SIMULATION FOR CPE USE)

THESIS

Daniel L. Petty
Captain, USAF

AFIT/GCS/ENG/84D-21

AFIT/GCS/ENG/84D-21

DEVELOPMENT OF A USER SUPPORT PACKAGE FOR CPESIM II (A COMPUTER SIMULATION FOR CPE USE)

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Daniel L. Petty, B.S.
Captain, USAF

December 1984

Preface

Dr. Thomas Hartrum provided the initial stimulus for this project. Previous thesis students had attempted to create for him a complete simulation package for use in his CPE classes but he was still unsatisfied with the results. When he offered me the opportunity to do some user friendly interfacing, I grabbed at the chance.

A previous assignment had given me some initial exposure to user friendly interfacing and I was grateful to Dr. Hartrum for offering me the chance to do more such work and on a more independent basis. My only regret is that required classwork took up time that I would rather have put into this thesis effort to make it do even more than Dr. Hartrum originally asked.

I encountered a classic problem while writing this document; the matter of gender. To constantly read "his or her" or, as I have often seen, "his/her", can soon become irritating, particularly in a lengthy document. With all due respects to the female of our species, whenever a masculine pronoun appears in the text, its use is generic; either gender can apply to the reference. If this convention offends anyone, they have my humblest and sincerest apologies.

I would like to thank Dr. Hartrum for the opportunity he gave me and all the guidance he provided. I would like

to thank Lt. Col. Thomas Clark (now retired) for his insights into user friendly interfacing. I appreciate the inputs of Maj. Walter Seward and am indebted to Maj. James Coakley for serving on my thesis committee. I also owe a debt of gratitude to my family for their constant and much-needed moral support. I have saved my greatest thanks, however, for my wife Tracy for her infinite patience when I neglected household duties and for her support when I worked far into the night, all due to this project. Her contribution, and that of Ms. Sharleen Hake (to whom I am also greatly indebted), in the final frenzied typing stage ensured the "timely completion" of this report.

Daniel L. Petty

Contents

The following is maintained at AFIT/ENG:

Volume II

List of Tables

Abstract

In 1983 a SLAM computer simulation was developed to be used as an educational tool in CPE and Queueing Theory classes at AFIT. Lack of user friendliness and additional requirements necessitated the development of a user friendly interface, i.e. this thesis effort. The interface consists of two programs: one for the instructor and one for his students.

The students' interface allows them to modify and display the initial or subsequently modified computer configurations and to do a data reduction and histogram analysis on output software monitor data after a simulation run. Any student configuration changes are stored in an Ingres database.

The instructor's interface allows him to easily create the initial configuration, the catalog of available hardware, and the system's workload of from one to ten jobstreams. He can also display any or all configurations of any or all student teams on his terminal or direct that display to a file for later printing.

The interface as a whole is menu-driven, user friendly, and very portable; it operates on any UNIX system (which has Ingres and SLAM) regardless of the hardware (including terminals) that the operating system is implemented on. A student and instructor user's manual is provided.

Development of a User Support Package for CPESIM II (a Computer Simulation for CPE Use)

I. Introduction

Background

Sometime during the lifespan of every operational computer system, the computer's workload exceeds the capability of the system as configured. Historically, the solution to this problem has been to simply add more hardware to the computer, or if the system is already at maximum hardware configuration, to buy a new computer system without a thorough analysis of the workload of the current system. Recently, however, more and more effort has been placed on analyzing the current system configuration to determine if additional hardware is actually needed or if the current configuration can be modified or "tuned" to handle the workload more efficiently.

In a 1972 study for the Air Force (1:6-9), Bell, Boehm, and Watson suggested seven phases for a computer performance evaluation (CPE) team to follow in successfully evaluating the computer system and its environment:

1. Understanding the system
2. Analyzing operations
3. Formulating performance improvement modifications

4. Analyzing probable cost-effectiveness of modifications
5. Testing the specific hypotheses
6. Implementing appropriate combinations of modifications
7. Testing the effectiveness of implemented modifications

To effectively learn CPE techniques, the student must master these seven phases. The first four can be mastered through classroom instruction. However, the last three phases require a more personal involvement and the student must have access to CPE data from a real or simulated computer system. CPE data usually includes hardware and software monitor data and accounting data. Since many computer installations can ill afford the price possibly incurred by allowing students to experiment with an operational system, a computer simulation is the optimal alternative. In 1979, Capt. Paul Lewis developed a SIMSCRIPT II model to answer this need (2:1-56).

Capt. Lewis' model, CPESIM, was instrumental in allowing students to test their CPE skills firsthand. Time, however, brought to light several deficiencies in the model. The most important of these was the fact the CPESIM's software monitor failed to provide the necessary information for the student to successfully analyze the problem. Also, due to the advent of the SLAM and QGERT languages, the language in with the model was written, SIMSCRIPT II, was no longer being taught at AFIT

and thus created a lack of "local talent" to sufficiently support it. Finally, the user interface was burdensome to both instructor and student, making the model's use unpleasant at best. In 1983, Capt. David Owen rewrote CPESIM in SLAM and provided embellishments to produce the proper software monitor data (3:1-35).

Problem

While Capt. Owen's model, CPESIM II, satisfied the first two of the aforementioned problems well enough, it was unable to interact with its users in a friendly manner. Since CPESIM II is intended as an educational tool, user friendliness is hardly optional. Educational tools should ease the task of solving or learning about a problem. If the tool is not user friendly, valuable time is diverted from the task to be solved or learned and is instead spent mastering the tool. User friendliness in a tool minimizes this diverted time.

Capt. Owen himself pointed out some areas in which CPESIM II was lacking. The generation of workload data was still being accomplished by the original SIMSCRIPT II program and that program could only generate a single job stream at a time. If the student wished to examine data from a multiple job stream workload, the workload generator program had to be run several times and the resulting files merged to form the required workload. Also, CPESIM II did little post processing of accounting or software and hardware monitor data

files. Finally, the process of tracking the changes a student makes to the simulated computer environment was a manual one. The student changes should be collected into a database to ease the instructor workload and to provide a "history" of the evolution of the student's final computer system configuration (3:38-39).

Scope

Since the kernal of CPESIM II was already operational, there were some limitations to the scope of this project. CPESIM II models a classic von Neumann machine and no modifications to this basic design were made. An analysis of CPESIM II was compared with the classroom needs of EE6.52 (Computer Performance Evaluation), EE7.52 (Advanced Computer Performance Evaluation), and EE6.47 (Queueing Theory), the primary users of CPESIM II, to determine all the requirements of the model. Those requirements which were not already integrated into the model were satisfied with additional modules or through modification of current modules. A total rewrite of the interface (such as it was) in another language was in order and a documentation manual for the new program was written for both student and instructor. Once developed, all new modules and programs were implemented, fully tested, and demonstrated to the user's satisfaction.

Assumptions

All projects which are accomplished stepwise have a common premise; it is assumed that all completed, underlying work is correct -- that it has been validated and verified. Since this project was a refinement of a previous one, that same assumption was made here. Also, it was assumed that the requirements of the three aforementioned classes were implementable. A final assumption was added in the early development phase of this project. It was at this time that an announcement was made by the Aeronautical Systems Division Computer Center concerning a billing change for time on the Cyber computer. Since the change would adversely impact this project if CPESIM II remained on the Cyber where it was originally developed, the sponsor decided that the entire project should be moved to the VAX/UNIX computer. This would enhance availability of the software to the students, eliminate the need for dependency on a new and tenuous communications link between the Cyber and the VAX (since the decision was already made by the sponsor to implement the interface on the VAX), and of course to avoid any economic crises imposed by the billing change. This decision was based on the important assumption that there is no significant difference bewteen implementations of SLAM on the Cyber and on the VAX -- that is, that equal simulations run on each implementation would produce equal results.

Approach

The first step in this project was to perform a complete analysis on CPESIM II to determine its full capabilities and its weaknesses. Then the requirements from the CPE and Queueing Theory classes were obtained by interviewing the current instructors of these classes. The requirements were compared to the current capabilities of CPESIM II in order to determine what modules needed to be added or modified. Each of the new modules were then developed through a modular top-down technique, namely Softech's Structured Analysis and Design Technique (SADT). Upon completion of the SADT development, an appropriate language that could meet the needs of the modules was chosen in coordination with the sponsor (who will be responsible for program maintenance upon project completion). At that point the actual coding of the modules began. Theoretically, coding should have been the shortest phase of the project, but historically speaking that is seldom the case (and was not here, either!). Next, each module was tested and verified for accuracy followed by testing and verification of the new model as a whole. Since some of the performance algorithms were changed between CPESIM and CPESIM II, it was not possible to verify the final product through parallel simulations. Therefore, verification was intuitive -- "best guess" on the sponsor's part. The only task left at this point was the writing of the student and instructor user's manuals. Upon completion of the manuals, the finished product was demonstrated to the sponsor, I believe to his complete satisfaction.

Order of Presentation

Chapter II describes the interface requirements in some detail. Chapter III outlines the interface design and contains a detailed discussion of thought processes and decisions which went into the interface design. Alternatives which were not selected in the design process are also discussed where appropriate. Chapter IV discusses the procedure which was used in testing the interface and provides some evaluation of its performance. The final chapter, Chapter V, provides a project summary and includes recommendations for improving the CPESIM II system. Appendix A provides a formalized top-down design structure utilizing Softech's SADT diagrams and Appendix B documents the problems and peculiarities the designer encountered with the C language and the Ingres database management system as implemented on the VAX/UNIX operating system. Appendix C contains relation definitions from the Ingres database storing the CPESIM II configuration and catalog data. Appendices D and E contain the student and instructor user manuals respectively. These provide the appropriate user with the necessary guidance needed to effectively use CPESIM II. Appendix F contains an abridged version of this thesis in publishable form. Volume II contains program listings for the interface and any modified simulation code.

II. System Requirements

Environment

The CPESIM II system is intended for use in graduate-level CPE and Queueing Theory classes. These classes focus on applying modern tools and techniques to evaluate an operating computer system. Specific case studies are presented throughout the term and the student is given case studies to which these techniques may be applied (3:7).

A requirement exists to give the student a real or simulated system in which he can apply the presented techniques to the given case studies. To successfully integrate CPESIM II with the case study, the instructor must develop a scenario describing the computer and its environment and either the specific performance problem or the desired goal. The description may be prosaic in form or may consist of only the raw monitor data and the accounting data of the system to be analyzed. The student analyzes the given data and, if he determines that the data is insufficient to allow him to propose a candidate solution, he specifies certain monitor parameters in the system's configuration and requests a simulation run. Once sufficient accounting and hardware and software monitor data has been accumulated by the student, he proposes a solution to the problem and implements it on the CPESIM II model for verification. Data from the next simulation run will verify whether or not the student has, in fact, solved the problem.

If not, the process of data collection, "hypothesizing", and candidate solution testing is reiterated until the problem is indeed solved (3:7-8).

Software Requirements

It was the primary goal of this effort to establish a user friendly interface to CPESIM II. To successfully accomplish this goal, it was necessary to meet requirements in two areas -- those of the CPE classes and the Queueing Theory class -- while still meeting the original system requirements (3:8-14). The sponsor also requested that the interface utilize a "generic terminal" concept; that is, that the interface would be able to run, without modification, on any UNIX system regardless of the terminal types available. This constraint tended to limit the variety of formats that the interface could have; complex graphics or other fancy terminal displays are generally somewhat terminal-dependent features of software. However, the advantage, beyond portability, is that the software can withstand any changes to the configuration of the computer that it is implemented on without having to be changed itself.

To satisfy the CPE classes' requirements, the interface must allow students to change the configuration of the simulated computer system and to elect whether or not to use the hardware and/or software monitors. Additionally, the student must be able to display the current system configuration or any previous week's configuration utilized by his student team, but should not be able to access any configuration of another

student team. As a group, students should be prohibited from modifying the standard equipment file and the system workload and they should be disallowed from initiating a simulation run (5).

The instructor, on the other hand, should be able to easily create the standard equipment file and the system workload of from one to a maximum of ten jobstreams and also the initial simulated computer configuration. Each of the parameters in the workload generator are described by one of a variety of the most often used (by computer simulations) frequency distributions. In particular, the sponsor requested that a user-defined discrete probability function be made available, so this was also included among the distributions which were "borrowed" from the SLAM simulation language. Since the developers of SLAM had already done a study of the most-needed frequency distributions (7:29-42), the CPESIM II interface designer felt that those distributions implemented in SLAM should cover the needs of CPESIM II users.

The instructor should also have totalitarian control over when the simulation will run and should be able to display any system configuration of any student team. Each configuration should be stored in a database to facilitate the quick display of any given configuration, past or current. Finally, the simulation must output raw software monitor data and the interface must do some post-processing of this output. The software monitor output should be collected and shown first as a distribution of the amount of time <u>all</u> jobs spent in

each queue, and second as a list, by individual job, of the time spent in the various queues (5).

Queueing Theory class requirements match well with those of the CPE classes. When presented with a scenario of a "real-world" computer system (much like CPESIM II's simulated system), the student must be able to describe the scenario with an appropriate mathematical model. To accomplish this end, the simulation must produce data concerning the number of jobs passing through the computer system and time delays encountered at each point. The interface must allow easy workload generation and configuration modification. The simulation requirements are already met by CPESIM II as it is. The interface requirements for the Queueing Theory class are closely in line with those of the CPE classes (6).

Support Requirements

The instructor is advised to create scenarios which force the student to make trade-offs; to give the student unlimited free rein is not realistic. The scenario should offer the student the total computer environment to aid him in learning the complexities involved in the decision-making process. Some constraints which might be introduced into the scenario include possible configuration limitations (due to space available or user preferences), a realistic cost budget (anyone can solve a performance problem given enough money to do it with), and intangible factors such as user and management needs

and goals (this will give the instructor the opportunity for some role-playing) (3:14).

The student will also require access to a standard statistical package (such as S or SPSS) in order to more easily and effectively analyze data from the simulation. Since the S package is available on the VAX, that package was selected to do the histogram analysis of the data in the monitor post processing program (3:15).

Verification and Validation

Since this effort was a program interface, with the exception of the software monitor data post-processing, any "verification and validation" merely involved checking that the implementation matched the design and performed without error. It took only the raw data and a bit of time to verify and validate a few pilot runs of the software monitor data processing. Examining the raw data and picking out all references to a particular job by hand and then reducing that data selected produced (after a great deal of time) the same results which were produced by the software monitor data post processor.

Summary

This chapter has reviewed only the high-level requirements for the CPESIM II interface. A more detailed look at the requirement is implicit in the SADT diagrams of Appendix A. If the reader feels the need for such detail, he is

directed to that source. The next chapter will discuss the design of the CPESIM II interface and include some detailed comments on how and why various design choices were made.

III. Interface Design

This chapter details the thought processes which shaped the design of the CPESIM II interface. If an alternative choice was examined and discarded, details of this process are discussed where possible.

Underlying Objective

There is an underlying objective which is the focus of the interface design; all software must be user friendly. This is an objective which may be obvious to some, but is not always inherent in a human-computer interface program. Without constant attention to user friendliness by the designer, the interface cannot fulfill its potential. Thus, this "obsession" for such a detail is not without merit.

Whenever possible, the interface has been designed so that it can be used independently of written documentation. Such "on-line" documentation speeds the learning process and furthers the objective of user friendliness. The designer felt that learning to use a new software tool should not be a process whereby the student plants himself before a terminal with the documentation propped up in view and engages in a repetition of reading a few lines of instruction, then entering some data via the terminal. Rather, the interface should prompt the student with easy-to-understand, natural

language messages so that minimal dependence on the user manual is necessary. The user manual should be a source of supplementary assistance rather than of primary instruction. This, then, was the overriding consideration throughout the design process.

The Language

The first choice to be made was that of the language to be used in the interface implementation. Obviously, it would have to be a language already available on the VAX in order to avoid unnecessary expenses and time delays due to delivery of a new compiler. Available languages were FORTRAN, Pascal, Lisp, and C. Of the four, C, FORTRAN, and Pascal were the most familiar to the sponsor (who would have to maintain the finished product).

Since the interface would have to interact with the user in as natural a way to the user as feasible, the chosen language would have to have superior string-handling capability. This requirement discounted FORTRAN as an alternative as its string I/O is rather unwieldy. FORTRAN 77 (the VAX implementation of FORTRAN) requires any string input to be enclosed in single quotes. This requires two additional keystrokes on the user's part. Since this would tend to increase the user's work unnecessarily (imagine entering 'Y' rather than Y for an affirmative response), FORTRAN was dismissed as an alternative.

The language would also have to allow complex data storage structures to avoid passing large numbers of arrays and control variables as arguments to subroutines. Both C and Pascal have this capability: C with its structures and Pascal with its records. Neither language had an advantage over the other in this respect, so no choice could as yet be made.

Finally, the chosen language would have to allow modularity in order to best support the application of good software engineering techniques. Both C and Pascal are considered modular languages; their "black box" approach to subroutine calls make the control of variable values much easier. Both languages were clearly superior choices for this application.

Having eliminated Lisp because of non-familiarity by the sponsor and FORTRAN because of its unwieldy way of handling string I/O, the choice of a language in which to implement the interface was narrowed to Pascal or C. The only advantage that C had over Pascal was that C executes somewhat faster on the VAX, but that was not an overwhelming consideration. A final decision still could not be made.

The Database

A choice of database type was the next consideration to be dealt with. An analysis of the configuration data would yield the ideal database type for this application. The configuration data required by CPESIM II is summarized in Table I.

TABLE I

Data Required by CPESIM II

CPU data
- Number of CPUs
- Name of CPU
- Relative CPU speed
- CPU cost
- Timeslice of CPU

Partition data
- Number of partitions
- Name of partition
- Size of partition (kbytes)

IOM data
- Number of IOMs
- Name of IOM
- IOM data rate
- IOM cost

Disk data
- Number of disks
- Name of disk
- Disk data rate
- Disk cost
- Disk connections to up to 5 IOMs

Tape data
- Number of tapes
- Name of tape
- Tape data rate
- Tape Cost
- Tape connections to up to 5 IOMs

Card reader data
- Name of card reader
- Card reader data rate
- Card reader cost
- Card reader connections to up to 5 IOMs

Printer data
- Name of line printer
- Line printer data rate
- Line printer cost
- Line printer connections to up to 5 IOMs

Software monitor data
- Software monitor desired
- Starting day, hour, minute, and second
- Stopping day, hour, minute, and second
- Queues to be monitored (up to 5)

Hardware monitor data
- Hardware monitor desired
- Starting day, hour, minute, and second
- Stopping day, hour, minute, and second
- Sample rate
- Timer probe connections (up to 2)
- Counter probe connections (up to 3)

Certain relationships were easily ascertained by surface inspection. The disk, tape, card reader, and printer data all shared four common data fields. They could be lumped together and treated as a group called peripherals. The software and hardware monitors each held essentially the same type of information. The major difference was that the hardware monitor required a sample rate. The timer and counter probes roughly corresponded with the five software monitor queues. Thus, the nine groups of data items could be reduced to six.

Closer examination revealed other relationships. For example, the CPU, IOM and peripheral data all have an associated unit cost. No matter how many of a particular model piece of hardware existed in the configuration, they would all have the same cost. Also, the IOMs and peripherals all have an associated data rate which is the same for all of a particular model piece of hardware. Similarly, all CPUs with the same model number will have the same relative CPU speed. These items are all characteristics of a particular piece of hardware and also appear in the hardware catalog. Since they are duplicated here, it was decided to include catalog data in the same database as configuration data. Thus, use of storage space would be optimized.

There is some data missing, however. While CPESIM II needs data concerning memory partitions, it does not seem to need data concerning memory hardware. Memory modules have a name, size, cost, and quantity that is not reflected in the data required for the simulation. Yet this data is

required to maintain the integrity of the system. If students are graded on their configuration partly on the basis of costs, they could conceivably have much more memory indicated by the sum of memory partitions than actually exists in the configuration as hardware (that is, the sum of the partitions exceeds the sum of the memory modules) unless some accounting of the hardware is kept in the database.

Since it is optimal to store as little redundant data as possible (in order to optimize use of secondary storage on the VAX), it is best to take advantage of these relationships among the data groups. A relational database system is based on this exact principle. In a relational database, data can be organized into "tables", or relations, to reflect the relationship one particular data type, or attribute, has to another. Entries, or tuples, in these tables store duplicate data only in some (but not all) of its attributes. If data duplication appears only in some of the keys, the database is said to be normalized.

To normalize relations, it is necessary to discern any functional dependencies on the attributes. In examining the data items, one can see that each piece of hardware has certain characteristics which define it. For example, a CPU has a model number, a name, a monetary cost, and a relative operating speed. If the CPU appeared in a list or catalog, the list would probably be organized on some key, generally the item's model number. Thus, a model number determines a particular

piece of hardware and that item is functionally dependent on the model number. In the example above, cost, speed and name are functionally dependent on model number. These dependencies are utilized to form the catalog relation where model number is the key attribute and speed, name, and cost are dependent attributes.

Similarly, other relations can be formed from the functional dependencies among the data items. If the number of identical memory modules is stored in a quantity attribute, then quantity is functionally dependent on model munber. By the same token, the quantity of partitions is dependent on partition size. If hardware items are identified with an identification (id) number, then CPU, IOM, and peripheral model numbers are dependent on id number. Monitor start and stop times and monitor connections are all dependent on the monitor type (hardware or software).

To put the relations in third normal form, a requirement for a relational database, it is necessary to make sure there are no transitive functional dependencies between the attributes in a relation. For instance, CPU speed and cost are functionally dependent on model number, but model number is dependent on id number. To put the relations in third normal form, these attributes must make up two separate relations. With this consideration in mind, the relations shown in Table II were developed. The CPESIM II database which was developed is not quite normalized in the strictest sense. The reason that it

TABLE II

The database relations for CPESIM II

Catalog

model ratesize name cost type

CPU

qtr team week idnum model timeslice

Memory

qtr team week model qty

Partition

qtr team week size qty

Password

qtr team pwd

Periph

qtr team week idnum model

IOM

qtr team week iomnum model

Connect

qtr team week idnum iomnum

Monitor

qtr team week type use startday starthr
startmin startsec stopday stophr stopmin stopsec
samprate con1 con2 con3 con4 con5

is not normalized is that the timeslice attribute of relation CPU will hold the same value for all tuples which have the same qtr, week, and team combination of values. This was done to reduce overall storage as this one piece of data did not warrant the creation of a new relation. The sacrifice of total normalization was made in favor of the reduced storage space achieved by merely tagging this attribute onto another relation. Thus, an optimal data storage mechanism has been achieved to store both the configuration and the catalog of hardware.

There is only one relational database system available on the VAX. The Ingres database management system (DBMS) is an interactive relational database which has an embedded data manipulation language interface to the C language. Since Ingres is the only relational DBMS available and it can only interface with C, by choosing the DBMS a choice of implementation language is also made.

Function Control

Having decided on an implementation language and chosen a database system to store the data, it became necessary to choose a construct whereby control of the CPESIM II system could be maintained. Recall that the sponsor wished to keep students from accessing another student team's configuration data. Ingres has no facility to accomplish this end, but the interface could easily do so by storing a password for each team in the database. Thus, before accessing any other data

in the database, the interface would check a student-supplied password against the stored password to evaluate his authorization to the information requested. This explains the existence of the last relation in Table II, the password relation.

This password concept effectively blocks one student team from gaining access to information belonging to another student team and thereby gaining an unfair advantage in a "graded competition." If the instructor wishes to allow students to access each other's data, he can make the passwords for all teams identical. However, this would allow one student team to alter the data of another team, and this should be discouraged at all costs.

While a password construct effectively blocks one student team from another, this is probably not an effective block for preventing student access to areas reserved for the instructor. Also, forcing the instructor to enter a password whenever he wishes to run a simulation or check on student progress is hardly desirable when he is the only instructor. Thus, a different protection scheme should be employed to prevent access to what should be his and his alone. The simplest course of action in this case was to take advantage of the security in the VAX file system itself and split the interface into two separate programs, one for the instructor and one for the students. Now it was possible to locate the two programs in different places in the VAX file system and deny permission to anyone but the instructor for the instructor program (called cpoctl), yet give global permission to the

student program (called cpesim). Now the requirements to block one student team from another and to prevent students from running the simulation or changing the hardware catalog or workload files have been effectively satisfied. Naturally, a bright student who has some knowledge of Ingres could bypass the interface and access the database interactively, but he would need to know the name of the database to do that. To prevent this, the instructor should not publish the name of his database.

The Students' Program -- cpesim

The student program, cpesim, performs three basic functions: to display a particular configuration, to change a particular configuration, and to post-process a software monitor file. Either of the first two actions requires a retrieval from the database if the configuration desired has not already been retrieved. When a configuration is changed, the interface determines the week number for the last configuration stored by that team and stores the new configuration as that for the following week. Thus, the student need not remember which week is the current week, only which week's configuration he wishes to change. Any previous week's configuration may be changed to become the current week's configuration. However the student must input all of his team's changes in one interactive session. Once a set of changes are made and saved, they are "cast in concrete" and cannot be further modified; students have only append access to the database.

Since the database is to provide a history of the student teams' evolving optimum system configuration, the reason for append-only access is readily apparent.

The cpesim program is menu driven. It was felt by the designer that this has become such a common and well-understood practice in interfaces that the learning process could be shortened by such an approach. This also furthers the objective of portability. A great deal of effort was expended to insure that the form of the menus is consistent throughout the interface in order the make the user as comfortable as possible in working with it. The menus attempt to be crystal clear in their meaning for the various alternatives, but there is always room for misunderstanding. The menus are hierarchically arranged, performing actual data manipulation tasks only at the lowest levels. Full utilization of C language capabilities are used with each menu to insure that only valid selections are input; bad inputs result in the screen being cleared and the menu redisplayed until a valid input is entered.

For the most part, any part of the configuration can be changed and in any order. The exception to this is that the student must add a memory module before he can add another partition if the addition of that partition would exceed the capacity of the current memory hardware. Cpesim keeps track of the total amount of partitioned memory and the total amount of hardware memory. At each partition change, the total amount of partition memory is displayed. If the student attempts to add a partition larger than the amount of unused

memory, he will be warned and informed of the amount of unused memory left.

This program was designed with the student in mind. Since the cpesim user will use the program only 10 or 20 weeks at most, the learning curve for it should be very short. Thus, as little dependence on the student manual (Appendix D) as possible was a major goal. Whenever the student decides to add or replace a piece of hardware, the applicable part of the catalog is displayed on the screen. The model number and name is considered sufficient and any further detail is left to the student manual.

The designer realized that it is very easy for the student to get carried away with his modification of the system configuration. For this reason, the student makes his changes to a second copy of the configuration in the program. If the copy gets changed incorrectly to the point that it would be easier to begin again, the program offers an abort option which discards the second copy and the student still has the original configuration copy intact. This seemingly strange procedure becomes crystal clear when one has to wait an excessive amount of time for an overburdened system to make a number of database retrievals to get the entire configuration. If the first copy of this configuration remains intact, then this retrieval need happen but once. When the student is satisfied with his changes and wishes to save them, they are saved to the primary configuration copy and then stored in the database; the second copy is discarded.

Certain "irregularities" occurred to make the interface design somewhat less than easy, as is often the case when two pieces of software written by separate, non-communicating parties are interfaced. The most cumbersome of these occurred with the "software queue to be monitored" and "hardware monitor probe points" selections. Capt. Owen used a sort of menu originally in his interface program and passed the value of the menu option to the simulation program. Thus, he "hard-wired" his menu into the code. While this in itself caused no problem, the fact that he left out choices four through ten on the hardware menu and five through ten on the software menu did cause considerable difficulty. It would cause the users no end of confusion to see a menu whose choices were not completely sequential; there would always be some doubt that the entire menu was being displayed and the belief that "something is wrong with the program." Thus, to interface directly, a conversion would have to be made. Any time the user saw the selections, either in the "change configuration" or the "display configuration" selection of cpesim, the selections had to be converted to the menu selection values that they had seen. Yet, when the values were stored in the database to be later extracted and fed to the simulation, they had to correspond to the numbers hardwired by Capt. Owen. Clearly, this is not very efficient, but time constraints precluded rewriting massive sections of the simulation to correct this problem.

The last option available to the student, post-processing the software monitor data, is somewhat of an "after the fact"

option. When the simulation has completed, the student may use this cpesim option to perform a data reduction and histogram analysis on the software monitor data. The student is not prompted for anything when this option is selected. In fact, it may appear that nothing has been done at all. What does happen is that a background job is spawned and that background job performs the required task. The data is collected and written as records, one record per job, which summarize how much time the job spent in each of the system queues. Then each queue is summarized graphically in a histogram reflecting the amount of time the various jobs spent in it. The histogram analysis is done by the S Statistical Package system. Histogram cells are hard-coded into the program and were selected on the basis of expected ranges for time in various queues. S is the only statistical package currently available on the VAX, an overwhelming factor in its selection for use here.

Overall, the cpesim program provides a fairly easily understood interface to the CPESIM II simulation. The menu structure tends to "lead the student by the hand" through the tasks he must perform to get the simulation data he requires. Once the student enters his "simulation requests" into the database, the instructor must use the cpectl program to execute those requests.

The Instructor's Program -- cpectl

While the instructor's portion of the CPESIM II interface, program cpectl, was designed to be very user friendly, it presumes much more foreknowledge on the user's part than the student portion does. After all, it is not likely that the turnover for CPE course instructors will be as high as that for CPE students. The instructor is presumed to understand what is required as input at the various points in the program where input is required, but every effort is made to "protect the instructor from himself." It is assumed that the instructor will use the interface for far longer than the student will and will thereby have more familiarity with it.

Although cpectl appears to the user to be one program, it is actually two distinct programs. The main portion is written in C and assists the instructor in generating the hardware catalog, running the simulation, listing the student teams' configuration histories, and generating the initial configuration. There is a subroutine, however, which is written in FORTRAN. This subroutine, jobgen, was written in FORTRAN to take advantage of certain random-number generating routines which exist in the IMSL system library. Rather than writing such routines from scratch and "re-inventing the wheel" so to speak, it was a vast time-saver to instead write a program in FORTRAN which could use the existing routines. IMSL routines are written in FORTRAN and, as such, are best utilized by FORTRAN programs. The function of jobgen is to generate a workload to be used by the simulation.

Subroutine jobgen can create a workload of from one to a maximum of ten job streams. Each job stream can, of course, have completely different characteristics. For each job stream, the instrutor must select a frequency distribution concerning job interarrival time, CPU time required for job completion, memory required (in 1K blocks) by the job, a relative job priority, the number of allocatable devices required by the job, the number of input cards in the job, the number of lines the job will output, the number of system disk blocks needed, and the number of allocated device blocks the job will need. Possible frequency distributions include exponential, uniform, Weibull, triangular, normal, lognormal, Erlang, gamma, beta, and Poisson distributions. The instructor may also define a step distribution if none of the aforementioned will meet his needs. By selecting one of these distributions to characterize each of the aforementioned data items, the instructor can create a rather large number of different jobstreams.

Another of the functions of program cpectl is to generate the hardware or equipment catalog. This is a rather simple procedure which involves a continuous iteration of entering a model number, name for the piece of equipment, data rate or size or speed or whatever associated quantitative measure is required, hardware type, and the associated cost. The iteration is ended when a model number of zero is entered.

An early approach to storing the catalog information did not have a type attribute. Instead, certain key names

were required to be placed in the name field. This was necessary to retrieve the appropriate catalog information when a piece of hardware was being added to the configuration. Recall that the written documentation was not to be a crutch, but a supplement; the on-line documentation was to be as helpful as possible. It was undesirable for the students to have to memorize the equipment catalog. But limiting the names to be given the pieces of equipment was immediately recognized as an undesirable approach. For one thing it was far too rigid. More importantly, that approach did not allow for a new type of equipment to be added to the catalog. For instance, if next year someone discovers how to store 300 trillion bytes of information on a dime sized crystal, a new peripheral (called a crystal drive?) would have to be added to the catalog.

Adding a new attribute, the type attribute, to the catalog relation proved to be the best answer. It allowed the inclusion of any conceivable device to any of five classes: CPUs, primary memory modules, add-on memory modules, IOMs, or peripherals. A device outside one of these classes could not be added anyway because the simulation would not be equipped to handle it. Thus, maximum flexibility is maintained through a relatively simplistic structure. The classes are given two character codes; cp for CPUs, m1 for primary memory modules, m2 for add-on modules, io for IOMs, p1 for disk drives, p2 for tape drives, p3 for card readers, and p4 for printers. With these coded types, retrieving, say, all peripherals

becomes a much simpler matter ('retrieve peripherals where type = "p?"' as opposed to 'retrieve peripherals where name = "disk" or name = "tape" or name = "card reader" or name = "printer"'!).

Another function of cpectl is generating the initial system configuration for the students to analyze. This is done in exactly the same way as the students make changes to configurations and with virtaully the same routines. The minor difference is that the instructor specifies which week number the initial configuration will have. Arguments can be made for this number to be zero or one and both arguments are equally valid. Although the initial week number can be anything, zero or one is probably the best choice.

Program cpectl also prints out the history of each team's evolving configuration. This allows the instructor to recap his students' performance, to determine where they made mistakes, if any, and to determine if and when they discovered the errors and what they did to correct them. It also gives him the ability to do so correctly, as the data, once stored, cannot be altered by a student. The instructor has the ability to print out all of a given team's configurations or any particular one of them singularly. This allows him to not only review the history as a whole, but to check periodically during the term on any team's work. This section of the program is virtually identical to the "display configuration" routine in program cpesim.

The final capability of program cpectl is the ability to initiate the simulation run. The instructor can initiate a simulation for all student teams or, if something has gone wrong with a single team's simulation in a previous run, a single simulation for any given team may be selected. In either case, there are several steps to this process. The instructor is prompted for the week number of the simulation run, the total number of student teams (or, alternatively, which single team's simulation will run), and the name of the file containing the system workload. The instructor is advised to create the workload prior to attempting to initiate the simulation run. While only three inputs are needed to begin the "initiate simulation" process, many steps are involved. The program first reads the SLAM network code from file sims and writes it to a file called siminX where X is the team number. Then cpectl retrieves the configuration of team X for the current week and appends it to file siminX. Finally, the workload file is appended to siminX and the file closed. Next the SLAM subsystem is called using file siminX as the input file to the simulation. The process is then repeated for all the other teams. The reason that the three separate entities are written to the same file is simple.

SLAM, being written in FORTRAN, is a relatively inflexbile system. It can recognize an input file and an output file, but any other filenames must be hard-coded into the discrete portion of the simulation run. Under normal circumstances this might cause no major problem, but there will be

several unique simulations running concurrently which were initiated by cpectl. If the configurations, for example, were written into file config, all simulations would be looking for a file called config. The UNIX operating system will not allow several files with the same name to exist in the same directory. The alternative would seem to be consecutive simulation runs. But these simulations are anticipated to take a great deal of both CPU time and wall clock time. Consecutive simulations are not a feasible alternative. It might be possible to use a tape file as input, but the SLAM subsystem is called by a UNIX system macro which would have to be changed to accomodate this procedure.

Fortunately, SLAM has a standard input file (which can be named at system level) which it internally calls "ncrdr." All SLAM network code is placed into this file as well as any other data the program may wish to read. Therefore, the network code, the configuration data, and the workload data can all be copied into a uniquely-named file and passed to SLAM without fear of conflict with a concurrent simulation. Furthermore, this method reduces the number of extraneous files in the system which would have to be later deleted.

At the termination of the simulation, all simulation output is written to a file name SIMOUTX where X is the team number. This is the data that the students will need to examine to determine their next step in optimizing the simulated computer's performance. The instructor should advise the student to move this file into his own area as the information

will be overwritten at the next simulation run.

Extensive modification of CPESIM II I/O was necessary to accommodate the new method of inputting configuration and workfile data. Previously, CPESIM II had hard-coded file names from which to input this data, but with the relocation of the simulation software to the VAX system and the potential for an overlarge number of extraneous files on the system some modification was warranted. Except for some relatively minor error fixes, no other modifications to the simulation programs were necessary. This is not to say that none should be made. Some rather gross inefficiencies exist that even Capt. Owen was apparently aware of (3:39). These should be streamlined in the interest of better overall performance.

IV. Verification and Validation

This chapter discusses the procedure that was used in testing the CPESIM II interface. The validation process and how it applies (or fails to apply) here will then be discussed. The reader is reminded to consider briefly the focus of this project (see "Underlying Objective", Chapter III) before proceeding in order to gain the proper perspective.

Verification

The testing of the interface programs was so straightforward and simplistic that it is barely worth mentioning. The approach was done on two levels. First, the places in the program which required an input from the user were tested. Most of these were menu selections. Due to the way the menus were implemented, it would be nearly impossible to get an invalid selection past them. The selections were tested anyway, first to see if an invalid selection could be made and then to see if a selection performed as intended. As expected, the menus performed flawlessly. The other places for input were the places where actual data was being input. As much as was practical, the data input was verified as valid before acceptance. For example, when the user wished to change the value of the use field from 'n' to 'y' or vice versa, the mere selection of that memu option toggles the value from one to the other. In this way, only a valid value can be in the use

field. Also, the connections between hardware items are represented graphically on the screen by ones and zeros. If the user enters anything other than a zero, it is transformed into a one. Thus, no bad values can appear here either. In most places, however, little editing of data values can be done. Therefore, the program relies on the ability and intelligence of the user to validate his own data. The program enhances this ability by allowing users to change any data item, including one which has previously (in the same interactive session) been changed.

The second approach was to test that the user friendliness objective had been achieved. This was accomplished first by programmer testing and then through testing by "disinterested third parties." About 20 classmates were asked to sit down at a terminal and try to use the CPESIM II system, without benefit of any written documentation, after only a two minute briefing on what they were trying to achieve and how to access the system. This test was passed with great success; even without any written documentation, the students were able to use the system intelligently and understand what it was they were doing. No questions were answered by the test monitor (the designer) while the students used the interface program. In spite of this, the students were able to figure out how to use the program in a very short time. It was not necessary to run the test for more than 20 minutes. Some valuable feedback from the test participants resulted in several cosmetic changes to the interface to promote understandability.

Having passed this phase of testing, a user was selected who was somewhat computer literate (the person was accustomed to using a word processor on a minicomputer system) and asked to take the same test under the same conditions as the graduate-level computer students. Once again, the test was easily passed in the 20 minute time period, even though this person had no idea of how computers are configured or what CPE is. Of course, the testee was not able to do any CPE analysis, but there was no difficulty encountered in trying to use the software.

Validation

It is not altogether certain that program validation applies in this case. An interface program either works or it does not work; it has only one basic function. The proof of whether or not the system is indeed user friendly will be, so to speak, in the pudding. That is to say that only through actual use over a period of six months or so will any problems or accolades come to light. The system as a whole could possibly be validated by comparing the input file created by the interface to the input files created under the old CPESIM II system while both systems are supporting the same scenario. If the simulation inputs are the same, the outputs should be the same. Unfortunately, the old CPESIM II system could not be reloaded onto the Cyber and made operational due to an operating system change since CPESIM II's original development, so this validation test was not made. Other testing, however, has turned up no potential problem areas.

V. Summary and Recommendations

Summary

This project has provided a user friendly interface to a previously unfriendly and cumbersome CPE simulation. The system as a whole provides the student with a chance to develop and hone his CPE skills by providing a computer environment in which he can freely operate.

The system allows the student to receive a scenario about a fictitious (or simulated real) computer installation which has one or more problems in providing reliable or effective computer support. The student has the ability to monitor computer operations and change the configuration of the computer, within any budgetary constraints imposed, if he so desires.

The instructor has the ability to create a scenario, a compatible computer configuration, a hardware catalog of available equipment, and any number of workloads that he wishes. He can track the student's progress and render any assistance or advice that he deems necessary. It also allows the instructor an escape from the frequent tedium of the classroom by providing a chance for some role playing. The interface greatly reduces the instructor's workload in utilizing this simulation in class.

Recommendations

Although the CPESIM II system is now fully operational, it is by no means complete in itself. There are several things remaining to be done which, although not worthy of a thesis effort, are perfectly suitable as independent study or class projects. The following enhancements are suggested:

1. Add a "help" capability to all menus and data entry points within the interface. If the user has any confusion at all as to what is being requested, he could respond "?" and receive a short description (printed from an external file) about what the program is looking for at that point.

2. Add a cost-tracking feature to the system. At present, the tools and structures for accounting for costs associated with many student actions or configuration changes are present in the interface program. Lack of time, however, prevented implementation of this nice-to-have feature. Any cost tracking must still be done manually.

3. Make the "clear screen" module terminal independent. Some effort was made in this direction (documented in Appendix B), but time constraints made abandonment of this path a necessity. The module should be able to determine what kind of terminal the user is working with, possibly by using the "termcap" UNIX utility or even by querying the user, and use the appropriate control sequence to clear the screen.

4. Change the simulation to allow boolean operations on the values at the hardware monitor probe inputs. This is a common enough practice in a real environment that the

capability should exist in the simulated environment. No research into this modification has yet been done.

5. Add new relations to the database to store workload parameter data and modify the cpectl program code to store and retrieve parameter data to the new relations. Currently, all workload parameters are entered every time a new workload is created. If the parameters are stored, the instructor need only enter the parameters he wants to change to create a new workload.

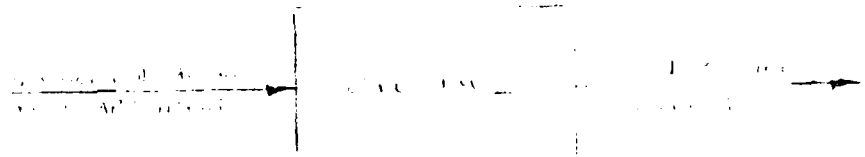
In addition to these enhancements to the interface, some work needs to be done on the simulation itself. For example, the simulation currently can only throughput 50 jobs before terminating abnormally. This is an unacceptable constraint to practical use of the system in the classroom. This problem must be resolved before the CPESIM II system can be implemented in the classroom. Also, the problem of the hard-coded menu values which were documented in Appendix B remains a burden to the interface. By changing the simulation so that the selections are sequential, the interface operation can be streamlined somewhat.

Appendix A

CPESIM II Interface

SADT Diagrams

AUTHOR:	DATE:	READER				
PROJECT:	REV:	DATE				



A-2

NODE:	TITLE:	NUMBER:

A0 CPESIM II

Abstract: This system allows an instructor to create a scenario within which CPE students may freely practice honing their CPE skills.

A1 Student Interface -- Performs all the functions which the student requires and the instructor allows within the instructor's scenario.

A2 Instructor Interface -- Performs all the functions necessary to allow the instructor to implement his scenario. Also aids the instructor in evaluating his students' performance.

A3 Simulation -- Simulates the computer system's operation within constraints provided by both instructor and student.

AUTHOR:	DATE: MAY	READER				
PROJECT:	REV:	DATE				

NODE:	TITLE:	NUMBER:

A1 Student Interface

Abstract: This activity performs all the functions which the student requires and the instructor allows within the instructor's scenario.

A11 Validate User -- Determines whether or not the user has proper access to the information he is requesting.

A12 Get Config -- Retrieves the computer configuration from the database and puts the data into an internal structure.

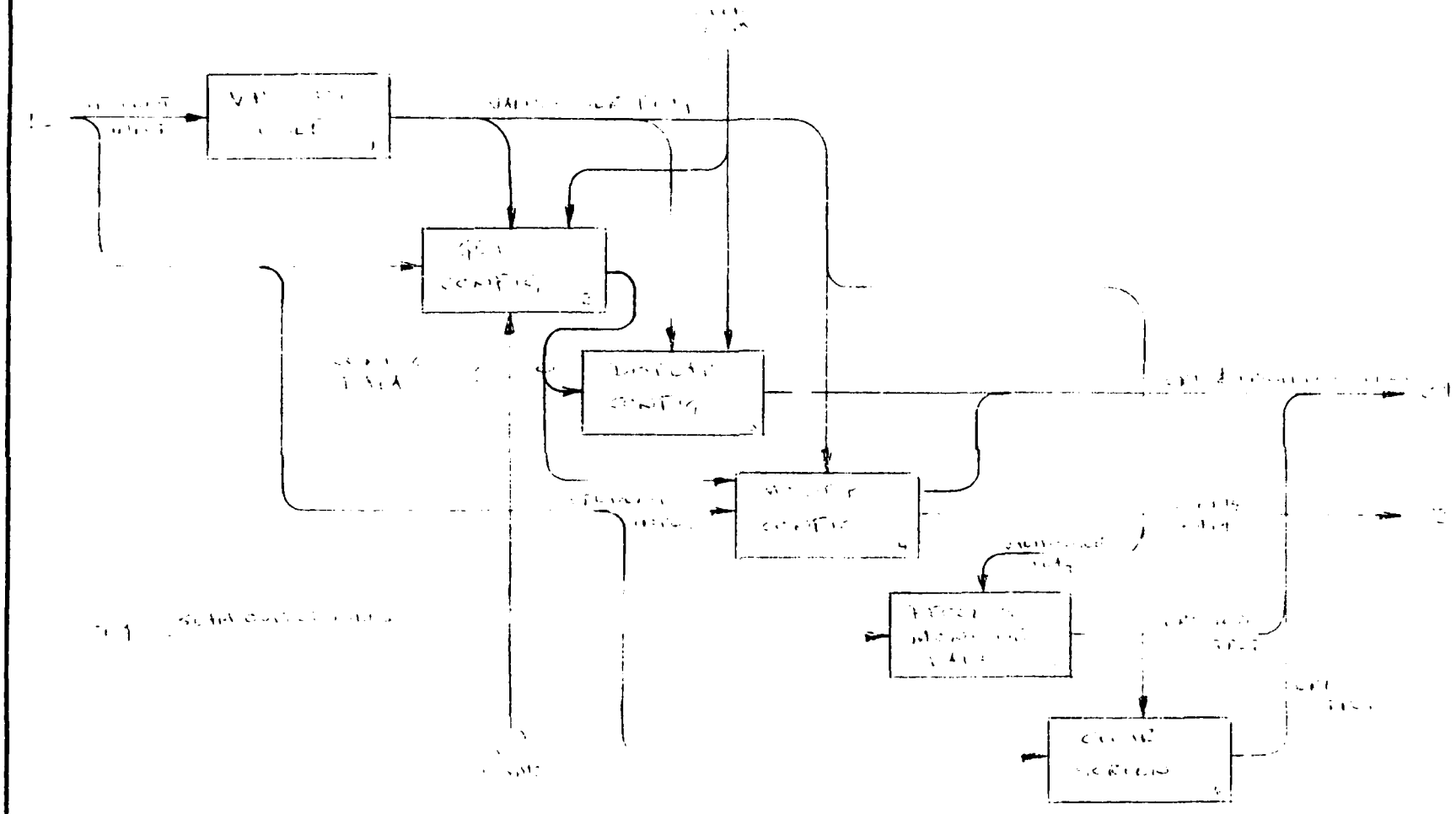
A13 Display Config -- Displays the retrieved configuration onto the screen or, in the instructor's case, into a file.

A14 Modify Config -- Accepts changes to the retrieved configuration from the student and, when all changes have been input, stores them in the database as a new configuration.

A15 Process Monitor Data -- Performs a data reduction and a frequency analysis on the software monitor data output from the simulation.

A16 Clear Screen -- Clears the user's terminal screen.

AUTHOR:	DATE: MAY 24	READER				
PROJECT:	REV:	DATE				



NODE:	TITLE:	NUMBER:

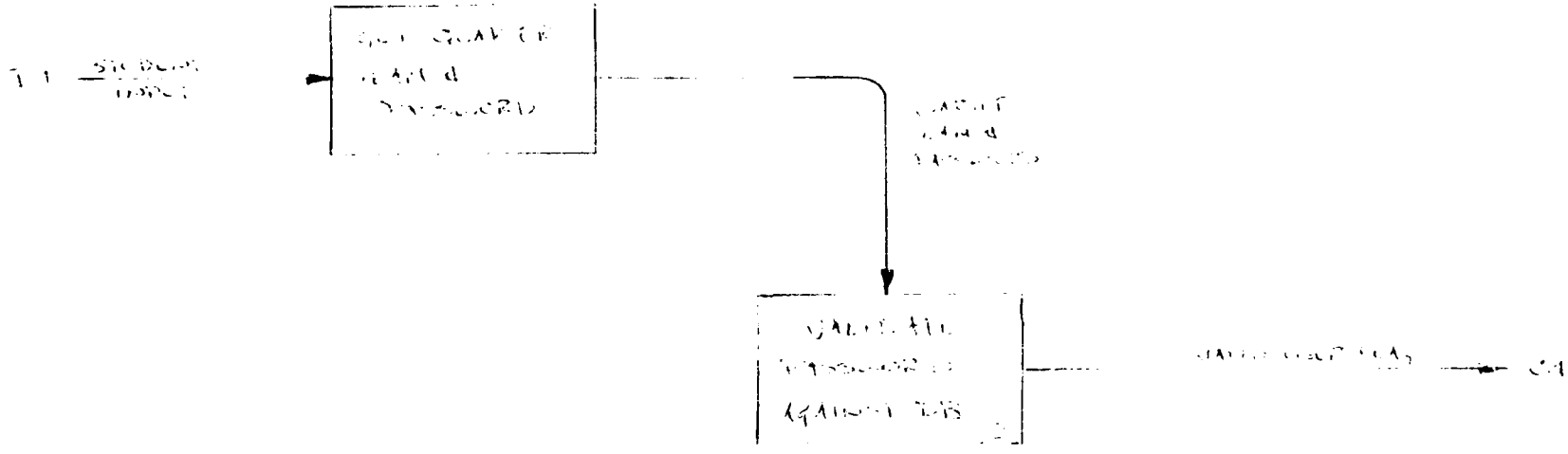
A11 Validate User

Abstract: This activity determines whether or not the user has proper access to the information he is requesting.

A111 Get Quarter, Team & Password -- Like it says, requests from the user the current quarter, his team number, and his team's password.

A112 Validate Password Against DB -- Checks the password input by the user against the password stored in the database. If the passwords are identical, the routine returns a value of 'true', otherwise returns a value of 'false'.

AUTHOR:	DATE:	READER				
PROJECT:	REV:	DATE				



NODE:	TITLE:	NUMBER:
A1	VALIDATE USER	

A14 Modify Config

Abstract: This activity accepts changes to the retrieved configuration from the student and, when all changes have been input, stores them in the database as a new configuration.

A141 Modify CPU -- Accepts changes to the system's CPUs.

A142 Modify Memory -- Accepts changes to the system's memory modules.

A143 Modify Partition -- Accepts changes to the system's memory partitions.

A144 Modify Peripheral -- Accepts changes to the system's peripheral devices.

A145 Modify Monitor -- Accepts changes to the system's use of hardware and software monitors.

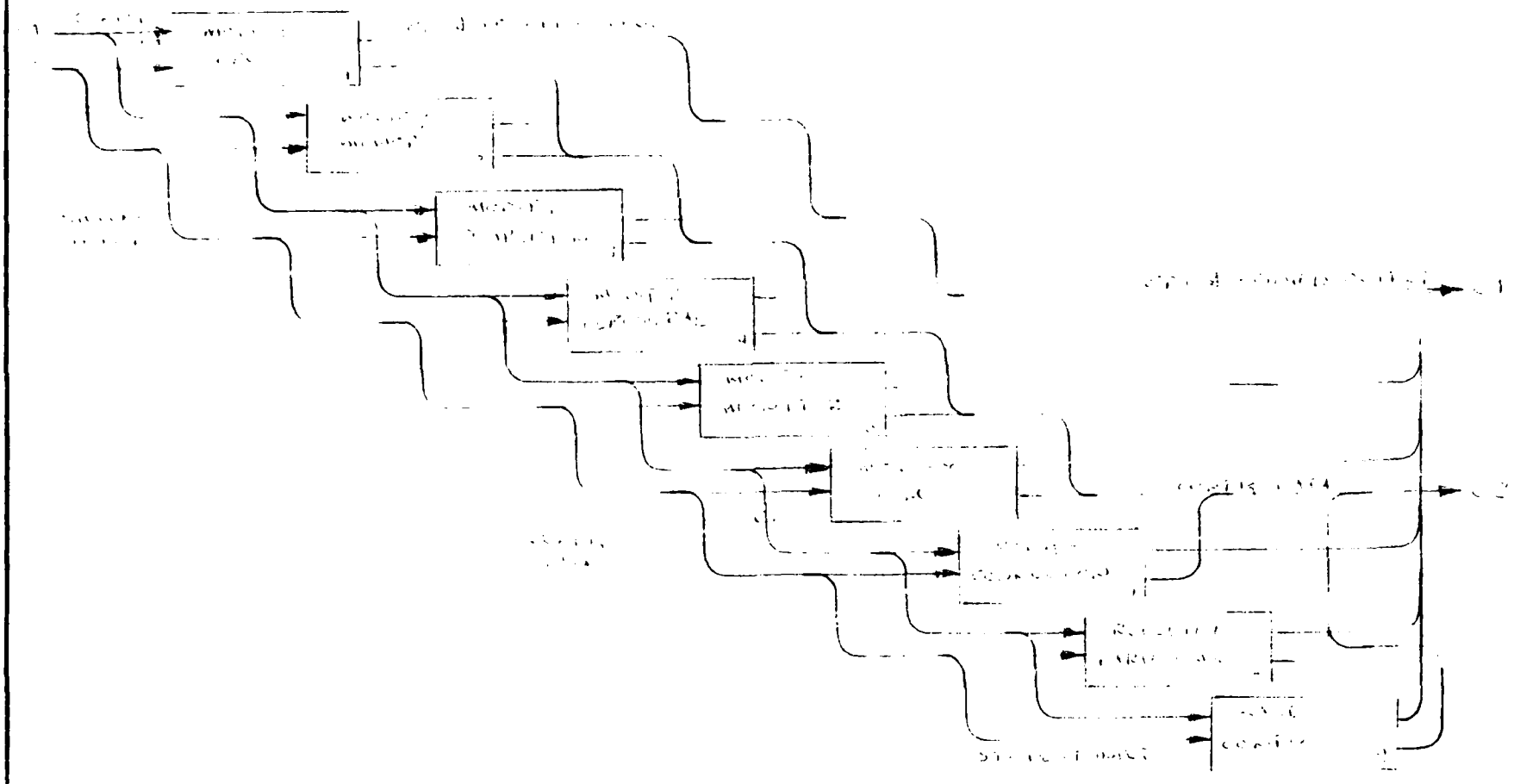
A146 Modify IOM -- Accepts changes to the system's IOMs.

A147 Modify Connections -- Accepts changes to the system's IOM-to-peripheral device connections.

A148 Redefine Partitions -- Enables the user to redefine all (vs. one at a time) of the system's memory partitions.

A149 Save Config -- Stores the new configuration to the database.

AUTHOR:	DATE: MAY 84	READER				
PROJECT:	REV:	DATE				



NODE:	TITLE:	NUMBER:

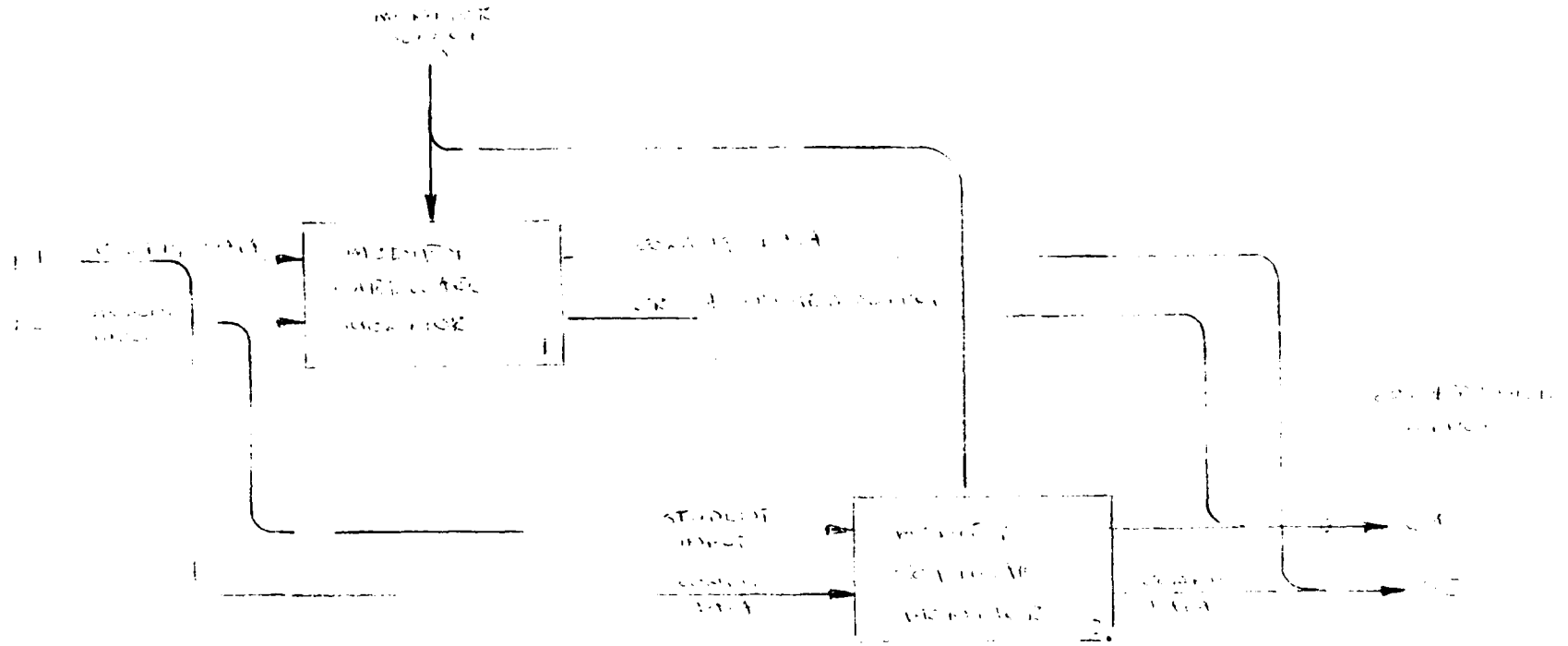
A145 Modify Monitor

Abstract: This activity accepts changes to the system's use of hardware and software monitors.

A1451 Modity Hardware Monitor -- Accepts changes to the system's use of the hardware monitor.

A1452 Modify Software Monitor -- Accepts changes to the system's use of the software monitor.

AUTHOR:	DATE: MAY 84	READER				
PROJECT:	REV:	DATE				



NODE:	TITLE:	NUMBER:

A2 Instructor Interface

Abstract: This activity performs all the functions necessary to allow the instructor to implement his scenario. Also aids the instructor in evaluating his students' performance.

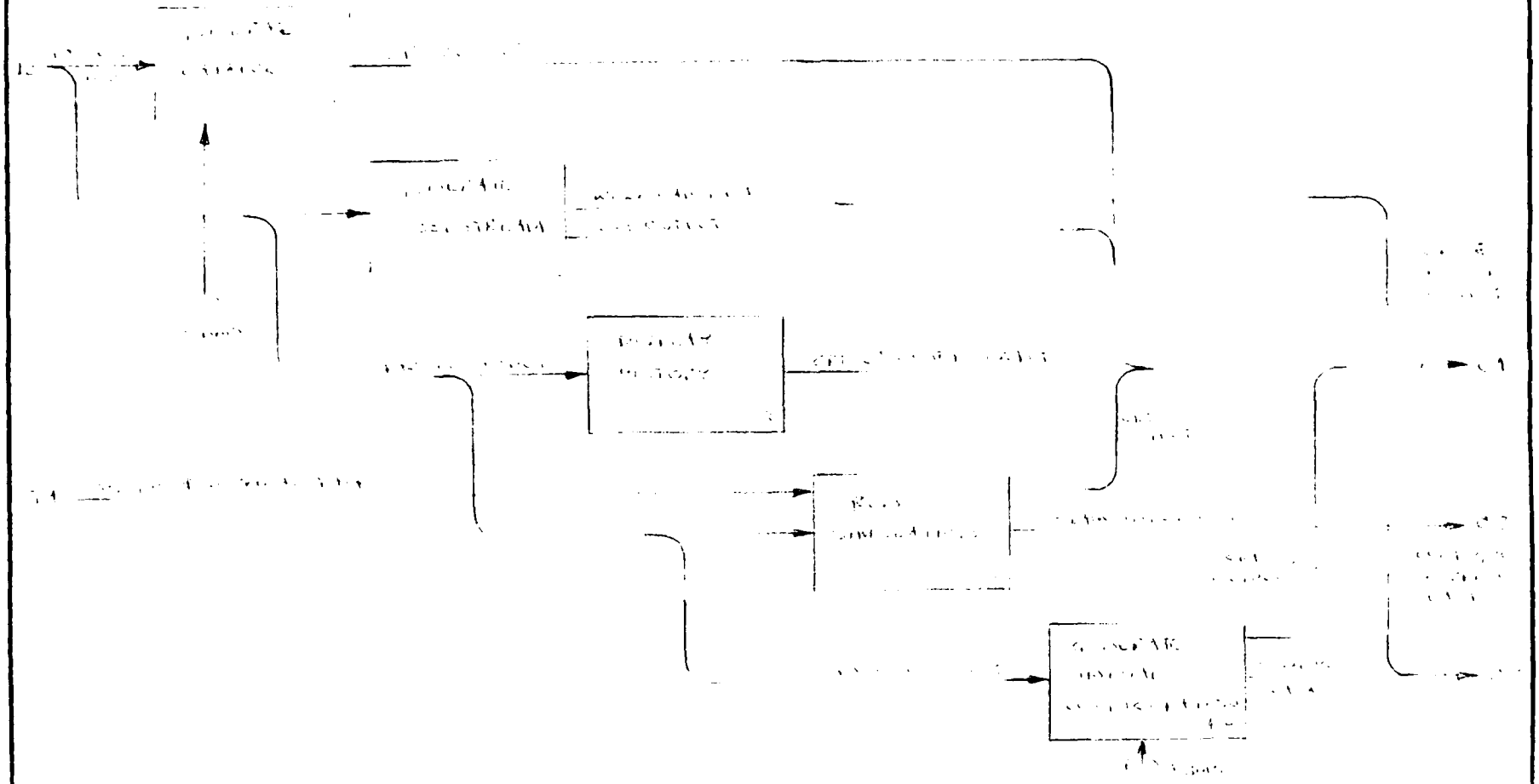
A21 Generate Catalog -- Accepts the instructor's entries into the equipment catalog and then stores them into the database.

A22 Generate Jobstream -- Accepts the instructor's parameters for describing the required system workload and then generates that workload.

A23 Display History -- Displays whichever team's configuration history (or single configuration) the instructor selects. He may also opt to see all configurations of all teams.

A24 Run Simulation -- Builds the SLAM input file from the SLAM network code, the configuration (which it retrieves), and the specified system workload. Unless the instructor opts to abort it, the simulation is then initiated.

AUTHOR:	DATE: MAY 84	READER				
PROJECT:	REV:	DATE				



NODE:	TITLE:	NUMBER:

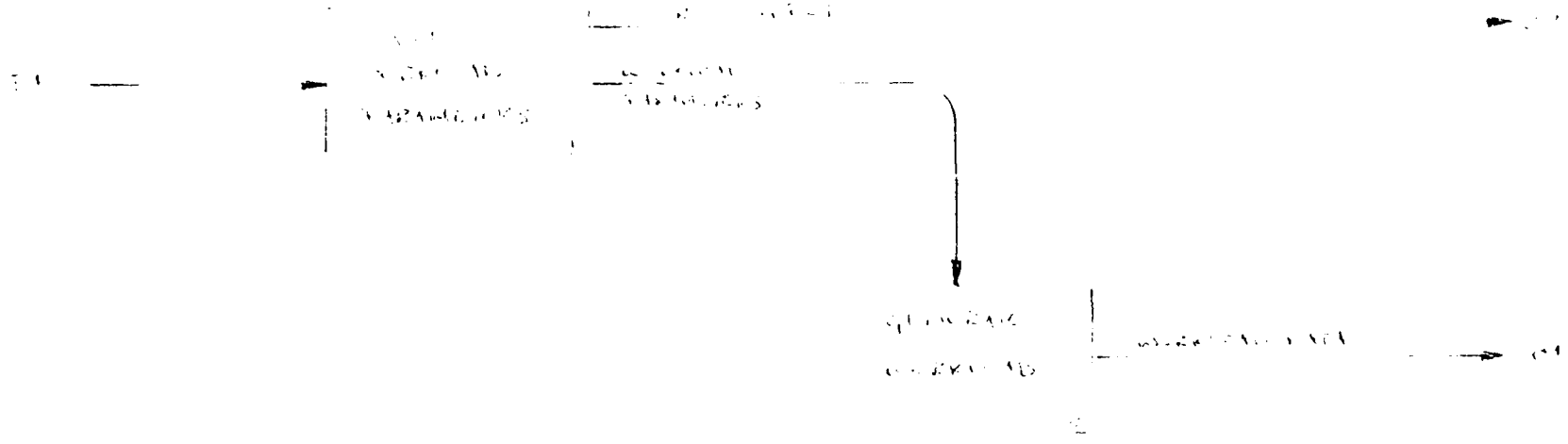
A22 Generate Jobstream

Abstract: This activity accepts the instructor's parameters for describing the required system workload and then generates that workload.

A221 Get Workload Parameters -- Accepts the parameters from the instructor concerning each attribute of the jobs from a particular jobstream.

A222 Generate Workload -- Accepts the parameters and actually generates the workload for the instructor.

AUTHOR:	DATE: MAY 84	READER				
PROJECT:	REV:	DATE				

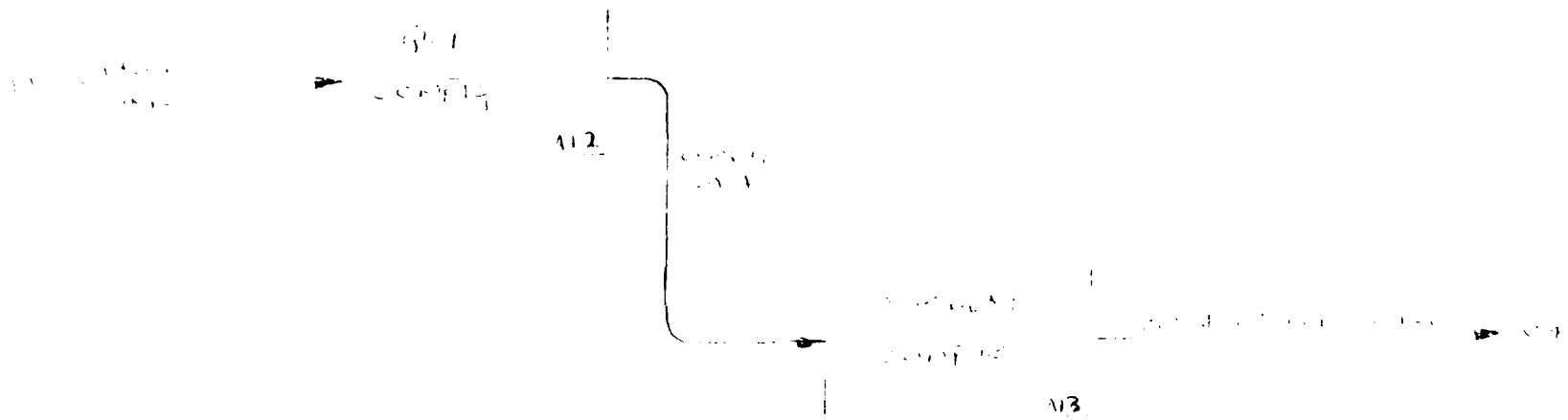


NODE:	TITLE:	NUMBER:

A23 Display History

Abstract: This activity displays whichever team's configuration history (or single configuration) the instructor selects. He may also opt to see all configurations of all teams.

AUTHOR:	DATE: MAY	READER				
PROJECT:	REV:	DATE				



NODE:	TITLE:	NUMBER:

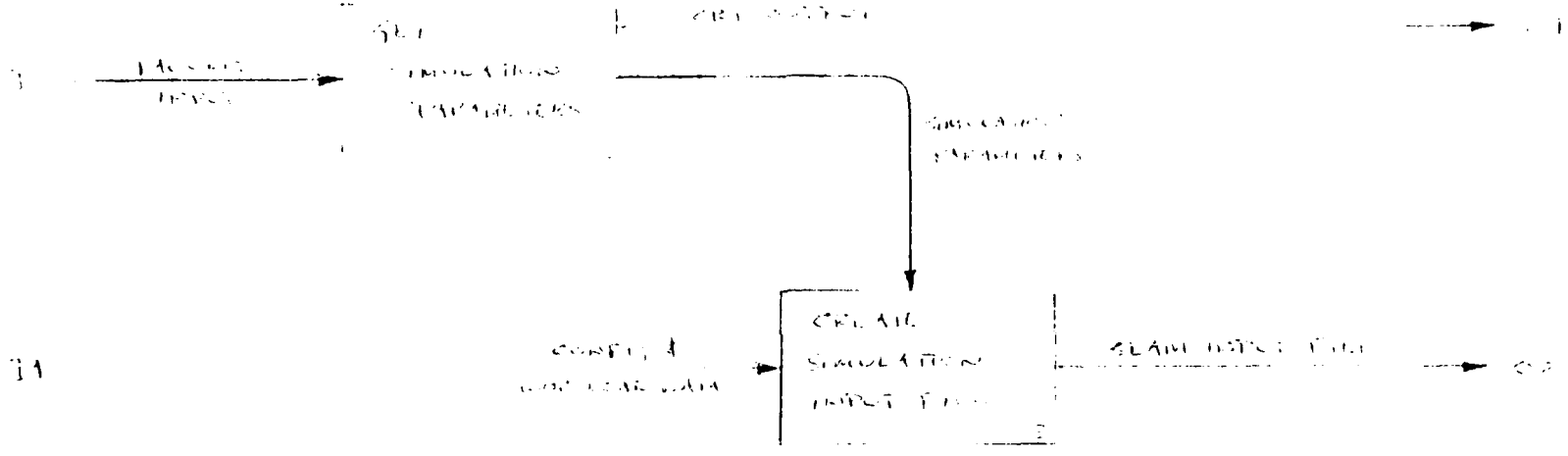
A24 Run Simulation

Abstract: This activity builds the SLAM input file from the SLAM network code, the configuration (which it retrieves), and the spedified system workload. Unless the instructor opts to abort it, the simulation is then initiated.

A241 Get Simulation Parameters -- Accepts from the instructor the quarter, week number, and how many teams for which the simulation will run as well as the workload to be used.

A242 Create Simulation Input File -- Creates the SLAM ınput file according to the input parameters and, if the abort flag is off, calls the SLAM subsystem to begin the simulation.

AUTHOR:	DATE:	READER				
PROJECT:	REV:	DATE				



NODE: A24	TITLE:	NUMBER:

Appendix B

CPESIM II Interface Implementation Problems

CPESIM II Interface Implementation Problems

Various problems with Ingres, the C language compiler, and the VAX/UNIX computer were encountered during implementation of the interface for CPESIM II. These problems are documented here in the hope that perhaps, being forewarned, other thesis students will be able to avoid them and the time loss they can represent to a project.

Ingres Problems

There are several problems, or at least undocumented limitations, with Ingres as implemented on the VAX/UNIX. First, the Ingres preprocessor often creates C code which, when compiled and executed, will give a bus error. There appears to be no good or consistent reason for this error. If fact, it is also possible that the error is the fault of the C compiler or even of the combination of the preprocessor's code and the compiler. At any rate, some minor shuffling of code, even something as minor as swapping the order of attributes in an Ingres target list, can cause the error to disappear.

When the designer of cpesim sought to avoid the above error by using a parameterized retrieve, he found an appalling lack of documentation on its use. The documentation, such as it is, discusses the use of parameterized quel statements (quel is the embedded C language interface to Ingres)

and gives an example of how to use them if one is not doing a parameterized retrieve to C variables. For the parameterized retrieve, the documentation says, "On a retrieve to C-variables, within [the target list], instead of the C-variable to retrieve into, the same '%' escape sequences are used to denote the type of the corresponding [argument vector] entry into which the value will be retrieved." This statement makes very little sense until one compares the example given for any parameterized quel statement (except the retrieve into C variables) with the example which should have appeared in the documentation for the parameterized retrieve into C variables. Both examples appear below.

The documentation example as included:

```
char *argv[10];

     argv[0] = &double_var;
     argv[1] = &int_var;
##   param append to rel
##       ("dom1 = %f8, dom2 = %i2", argv)
```

The example that should also have been included:

```
char *argv[10];

     argv[0] = &double_var;
     argv[1] = &int_var;
##   range of r is rel
##   param retrieve
##       ("%f8 = r.dom1,%i2 = r.dom2", argv)
```

The procedure for using a parameterized retrieve into C variables is now made very clear by the inclusion of the proposed example. The documentation as it stands leads the user to the procedure used by the cpesim designer -- trial and error.

Another problem with Ingres is that it does not sufficiently document its limitations. For example, the documentation specifies that, "A relation cannot have more than 49 domains and the tuple width cannot exceed 498 bytes." What the documentation does not say is that the preprocessor cannot handle a single Ingres statement which is longer than about 220 characters (no tests were done to definitively pin down the exact figure; the number 220 is within about 15 characters of the correct figure). This means that the user will be unable to retrieve all the attributes of a relation in a single retrieval unless the relation has far less than the limit of 49 attributes. Worse, the user would certainly be unable to store such a large tuple from inside his program as all attributes would have to be specified at once. A more realistic limit for a relation is about 14 attributes; a few more if the names are very short, a few less if the names are more than about five characters each.

The Ingres proprocessor, equel, is capable of producing bad code. For example, the source lines

```
## param append to cpu
##("qtr=%s,team=%i4,week=%i4,timeslice=%i4,idnum=%i4, model=%i2",
## argv)
```

when processed by equel become

```
IIwrite("append cpu(qtr=");IIcvar(qrtr,3,0);IIwrite(",team=";
    IIcvar(&tm,6,4);IIwrite(",week=");IIcvar(&weak,6,4);
IIwrite(",timeslice=");IIcvar(&time,2,4);IIwrite(",idnum=");
    IIcvar(&j,6,4);IIwrite(",model=");IIcvar(&stopd,1,2);
,2,4);IIwrite(")");IIsync(0);
```

The underlined code is a fragment from the previous line of code and is the cause of the compile error (when compilation is attempted). The only apparent remedy to this situation is to repair the preprocessor's bad code with an editor before attempting a compilation. The appearance of this error is sporadic.

Finally, equel statements to Ingres can take a very long time to execute, especially when the system user load is high. When the user load is light, interaction with Ingres seems to take almost no time at all -- on the order of five seconds or so. But when the user load is heavy, a configuration retrieve can take 5 to 10 minutes, a very long time to look at a terminal screen which is doing nothing. The Ingres user is advised to keep his database accesses to a minimum.

C Language Problems

Some problems also exist within the C compiler. A program which is by all appearances error-free can nonetheless compile and execute with a bus error. By shifting the order of variable declarations, this error can be made to disappear. The error occurs most often when character

arrays are declared before integer variables or integer arrays. Apparently, the character arrays force the integer variables off of full-word boundaries and storage problems arise. The C language programmer is advised to always declare floating point variables first, long integers, and character arrays. This ordering tends to avoid this problem.

Another problem which exists is a lack of documentation on long and short integers. Kernighan and Ritchie document the storage requirements for these types in their book "The C Programming Language" for the PDP-11, Honeywell 6000, IBM 370, and Interdata 8/32 computers, but not for the VAX computer. The cpesim designer has discovered that VAX storage requirements for these types most closely resemble those of the IBM 370. A short integer requires two bytes while a long variable requires four bytes. However, unlike the PDP-11, if the programmer declares a variable as 'int', the VAX will give it four, not two bytes of storage. This is relevant when the programmer is utilizing parameterized statements in Ingres, where the variables' storage format must be described. Apparently, expecting uniformity among Digital Equipment Corporation compilers is an act doomed to some disappointment.

VAX Computer System Problems

The usage of the phrase "VAX computer system" here implies not only the central hardware itself, but also the terminals attached to it in room 133 of building 640; the

reference is to the AFIT Engineering configuration of the VAX/UNIX system. Severe user loads encountered during implementation and testing of the CPESIM II interface helped reshape the interface design somewhat when it was realized what possible time delays could be encountered during student execution of the interface. The extreme user loads forced "turn-around time" on the system to be extremely long. Thus it was realized that, in view of the slow speed on Ingres previously mentioned, an economy of Ingres calls was in order. Also, since a configuration retrieval could, at extremes, take minutes rather than seconds, any changes being made to a configuration should be made to an auxiliary copy of the configuration. Once all changes were complete, the auxiliary copy would replace the primary copy and then the new configuration would be stored to the database. If changes were made to the primary configuration copy and the student had made sufficient errors to warrant starting over again, it would be necessary to retrieve the configuration from the database again, the current copy would no longer be intact. This "double copy" strategy insures the integrity of the original configuration when a gross number of errors are made and negates the requirement for duplicate retrieval of the same data.

Another problem area was with the VT100 terminals. The CPESIM II interface was to be as generic as possible in design to facilitate portability among UNIX systems. Unfortunately, the VT100 buffer is apparently very small. The

original design had a "clearscreen" function which just scrolled the old display off the screen and printed the new display. At 9600 baud, the VT100 buffer would lose half the display being sent it. It is very difficult to pick options from a menu whose middle is missing. To remedy the situation, VT100-specific control sequences were used to clear the screen. However, this impacts upon the desire to make the code as generic as possible. If the interface is to be moved from the VAX/UNIX system to another, changes may be required in this module.

Appendix C

CPESIM II

Database Relation Definitions

The Ingres Database

At the heart of the CPESIM II interface is the Ingres database where the catalog and all the student teams' configuration data is stored. Program cpesim is written to expect the database to be named cpedata, but if the instructor wishes to change the name, there is only one source code line that need be modified. The instructor should look for the line

```
## ingres cpedata
```

and change the name to whatever he wishes.

The relations of the Ingres database must not be altered in any way from the way they have been originally implemented. The instructor is advised not to attempt to rename any relations or to change the way the domains have been typed. To do so involves more than casual modification of the source code. In order to insure that the instructor has created a database compatible with the cpesim program, the following relation definitions have been extracted from the Ingres database which was created and populated to test the original program implementation.

```
Relation:                   catalog
Owner:                      dpetty
Tuple width:                32
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           18
Storage structure:          ISAM file
Relation type:              userrelation


 attribute name      type   length   keyno.

 model                i        2        1
 ratesize             f        4
 name                 c       20
 cost                 f        4
 type                 c        2
```

```
Relation:                   connect
Owner:                      dpetty
Tuple width:                8
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           50
Storage structure:          ISAM file
Relation type:              user relation


 attribute name      type   length   keyno.

 qtr                  c        4        1
 team                 i        1        2
 week                 i        1        3
 idnum                i        1        4
 iomnum               i        1        5
```

```
Relation:                   cpu
Owner:                      dpetty
Tuple width:                13
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           10
Storage structure:          ISAM file
Relation type:              user relation


 attribute name      type   length   keyno.

 qtr                  c        4       1
 team                 i        1       2
 week                 i        1       3
 idnum                i        1       4
 model                i        2
 timeslice            f        4
```

```
Relation:                   iom
Owner:                      dpetty
Tuple width:                9
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           15
Storage structure:          ISAM file
Relation type:              user relation


 attribute name      type   length   keyno.

 qtr                  c        4       1
 team                 i        1       2
 week                 i        1       3
 iomnum               i        1       4
 model                i        2
```

```
Relation:                   memory
Owner:                      dpetty
Tuple width:                10
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           10
Storage structure:          ISAM file
Relation type:              user relation

 attribute name      type   length   keyno.

 qtr                  c        4        1
 team                 i        1        2
 week                 i        1        3
 model                i        2        4
 qty                  i        2
```

```
Relation:                   monitor
Owner:                      dpetty
Tuple width:                33
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           10
Storage structure:          ISAM file
Relation type:              user relation

 attribute name      type   length   keyno.

 qtr                  c        4        1
 team                 i        1        2
 week                 i        1        3
 type                 c        1        4
 use                  c        1
 strtday              i        2
 strthour             i        1
 strtmin              i        1
 strtsec              f        4
 stopday              i        2
 stophour             i        1
 stopmin              i        1
 stopsec              f        4
 samprate             f        4
 con1                 i        1
 con2                 i        1
 con3                 i        1
 con4                 i        1
 con5                 i        1
```

```
Relation:                   partition
Owner:                      dpetty
Tuple width:                11
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           20
Storage structure:          ISAM file
Relation type:              user relation

 attribute name      type   length   keyno.

 qtr                  c        4        1
 team                 i        1        2
 week                 i        1        3
 size                 f        4        4
 qty                  i        1
```

```
Relation:                   password
Owner:                      dpetty
Tuple width:                13
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           5
Storage structure:          ISAM file
Relation type:              user relation

 attribute name      type   length   keyno.

 qtr                  c        4        1
 team                 i        1        2
 pwd                  c        8
```

```
Relation:                   periph
Owner:                      dpetty
Tuple width:                9
Saved until:                Tue Dec 31 00:00:00 1985
Number of tuples:           45
Storage structure:          ISAM file
Relation type:              user relation
```

attribute name	type	length	keyno.
qtr	c	4	1
team	i	1	2
week	i	1	3
idnum	i	1	4
model	i	2	

Appendix D

CPESIM II

Student Manual

Contents

List of Figures

List of Tables

CPESIM II
Student Manual

I. Introduction

CPESIM II consists of a major computer system simulation and its operating environment. It was designed as a laboratory aid to allow students to apply computer performance evaluation (CPE) tools and techniques to a hands-on computer system. The use of an actual operating computer system by students for such CPE studies is impractical for several reasons. First, the overhead required and the disruption caused by some measurement tools is prohibitive to use in an actual computer installation. Second, if measurement data is available, it may not be academically useful; the system may be operating as it should and therefore providing no problem-solving opportunities, or it may suffer under several interacting problems which, although realistic, cannot practically be resolved by a student as a one-quarter course project. Finally, if there is enough data to permit the student to do an analysis and propose a solution, an operating computer installation is unlikely to allow the solution's implementation or, alternatively, to be able to choose from several conflicting solutions which were proposed by the student teams. CPESIM II solves these constraints by providing a simulated environment in which the student can

gather whatever data is desired, analyze it, recommend and implement a feasible solution, and finally verify the effectiveness of the modified system.

CPESIM II consists of two parts. The heart of CPESIM II is a computer simulation (written in SLAM) of a large-scale computer system which executes on a host machine -- the VAX system. Because it is a simulation, there are many simplifications and limitations which restrict its representation of reality. The second part of CPESIM II is a simulated operating environment for the computer system simulation. This includes a computer-generated workload, a written scenario, budgetary constraints (if desired by the instructor), and system data in a form that can be manipulated and analyzed by the student, who is playing the part of a CPE analyst and/or a data processing manager.

The written scenario provides the student with operating details about the particular computer installation he is to investigate. The instructor configures the hardware, operating system, and workload to illustrate a particular problem or situation. Measurement data is then provided as requested to the student for analysis and decision-making on a periodic (e.g. weekly) basis. In order to force the student analyst to make tradeoffs (a real-world constraint), only a subset of the available data can be accessed during a given period.

CPESIM II Output to Students

Many types of information are available to the student to aid in the analysis. Some of these are free and automatic; some must be specifically requested. There is a cost associated with many, while some are mutually exclusive. The student must choose what data he wants and how he wants to use it. This section briefly describes the outputs available and directs the student to further information where it is available.

Manufacturer's data -- Literature of varying value which describes hardware and software features is available in printed form.

Installation documentation -- In-house documentation of the local configuration, parameter values, modifications, problems, etc. may be available in printed form. These are specific to a given problem and will be provided, if available, as a separate handout.

Accounting system data -- The computer's accounting system files are available at no cost to the student in a file on the VAX named ACTLOG followed by the student's team number (e.g., ACTLOG2 for team 2). Such data is in raw form and the format is defined in the manufacturer's literature for the appropriate operating system.

Hardware monitor -- A hardware monitor exists for

the simulated computer, but may have to be purchased or leased. Particular probe points, sampling periods, and sampling frequency must be specified by the student. As with a real hardware monitor, its use has no effect on system performance. The output data is available in a file on the VAX named HWMON followed by the student's team number.

Software monitor -- A software monitor is available for the simulated computer, but, like the hardware monitor, may have to be purchased or leased. As with real software monitors, this monitor requires memory, runtime, and system overhead to operate. Output data is available in a file on the VAX named SWMON followed by the student's team number.

Student Inputs to CPESIM II

One of the advantages of a simulation is that students can easily make radical (or not-so-radical) changes to the system. Thus, there are a number of inputs that the student may (or must) provide to CPESIM II as well as some that he cannot. This section briefly describes the CPESIM II inputs available to the student.

Workload -- The student has no control over the workload at all. The instructor controls the workload and may choose to vary it as often as he desires

or in response to situational developments (e.g. in response to a student-initiated "user education program").

Interview requests -- Student analysts may submit written questions to "installation personnel." The questions, however, may or may not be answered.

Monitor purchase -- Student analysts may submit purchase orders for the purchase or lease of any available hardware or software monitors.

Monitor input parameters -- If a monitor is being used, students must provide to the simulation a starting and stopping time for monitor operation. The software monitor, in addition, needs to know which queues are to be monitored, while the hardware monitor needs to know where the probes are to be connected and what the sample rate will be. These parameters are input while making any other changes to the system configuration prior to simulation run time.

Reconfiguration requests -- Students may direct reconfiguration of the existing hardware and operating system parameters. Any degradation of system performance, however, will be frowned upon by installation management.

Purchase of system options -- Additional hardware or operating system modules may be purchased and installed by submitting an appropriate purchase order. Information on current cost and availability will be

provided with each particular project.

Personnel hiring -- Requests for hiring additional personnel (e.g. for an additional shift) or for overtime authorization may be submitted.

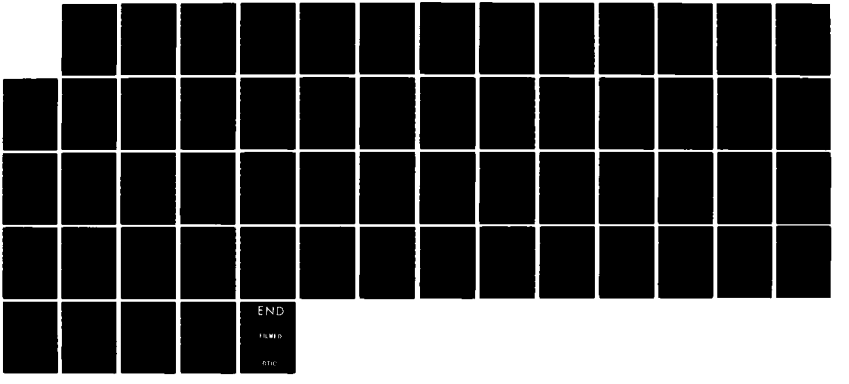
II. American Business Computer (ABC) BITBUCKET Computer System

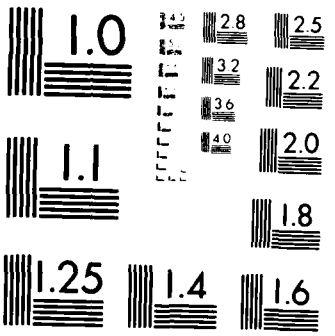
Introduction to the ABC Computer

The ABC computer system was designed to provide flexible, custom computer power to any organization. This system consists of a variety of hardware and software options which can be configured in many ways to tailor the BITBUCKET to the computing needs of the individual installation. This document provides abbreviated specifications of both the hardware components and the software options available to the ABC customer.

AD-A151 899 DEVELOPMENT OF A USER SUPPORT PACKAGE FOR CPESIM II (A COMPUTER SIMULATIO..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. D L PETTY

UNCLASSIFIED DEC 84 AFIT/GCS/ENG/84D-21 F/G 9/2 NL





Hardware Specifications

Model 2000 CPU

The model 2000 represents ABC's standard central processing unit. The Model 2000 provides for an extensive instruction set with an average execution speed of 250,000 instructions per second (IPS).

Model 3000 CPU

The Model 3000 provides essentially the same capabilities as the Model 2000 except that it operates at a faster speed. The Model 3000 has an average execution speed of 333,333 IPS.

Model 4000 CPU

The Model 4000 CPU is considered the crown jewel of ABC computing power. The Model 4000 provides the same instruction set as the Model 2000 and the Model 3000, but has an average execution speed of 500,000 IPS! The Model 4000 is truly state-of-the-art hardware!

Model 20 Core Memory

The primary main memory unit for the BITBUCKET system is the Model 20 Memory Module. This multi-port, 96K word module permits connection to up to 5 IOMs and up to 99 CPUs. The module can be expanded using Model 21 and/or Model 22 Add-on Memory Modules to have up to 20 partitions of 1000K

words each.

Add-on Memory Modules

Model 21 and Model 22 Add-on Memory Modules provide essentially the same capability except that a Model 21 Memory Module adds 32K words of core memory, while a Model 22 Memory Module adds 64K words of core memory to a Model 20 Core Memory.

ABC Input/Output Modules (IOMs)

The ABC IOMs are sophisticated I/O channels, each capable of interfacing several peripheral devices to main memory.

Model 3110

Model 3110 is a byte-multiplexed channel capable of multiplexing up to ten devices, each with a maximum transfer rate of 2.5K bytes per second. The primary function of this IOM is to provide an overlapped interface for the system's printers and card readers.

Model 3117

Model 3117 is a high-speed multiplexor channel capable of multiplexing block-oriented devices such as magnetic tape drives, disk drives, and drums. Up to ten high-speed devices can be connected to each Model 3117. Each device can transfer data at a rate of up to

500K bytes per second. The IOM itself is capable of transferring 500K bytes per second and can multiplex the transfer of multiple blocks as long as the sum of the data rates of the devices concerned are within the transfer rate of the IOM. This IOM requires interconnection to an accessed disk drive or drum during speed and rotational latency. In the case of high-speed drums, the IOM is dedicated during the entire delay period. For slower disk drives, the wait time can be multiplexed between several drives as long as the total transfer rate of the connected devices is less than the maximum transfer rate of the IOM.

Model 3119

The Model 3119 has all the capabilities and restrictions of the Model 3117 except that it can multiplex up to 15 devices.

Model 2700 Drum

The Model 2700 Drum has a fixed read/write head for each of 25 active tracks. It has a rotational speed of 3600 revolutions per minute and a transfer rate of 410K bytes per second. The storage capacity for each track is 7K words.

Model 2714 Disk Drive

The Model 2714 Disk Drive consists of a controller and one single-spindle disk drive which connects directly to the

IOM. The storage capacity of each mounted disk pack is 20.48 megabytes. Each drive provides access to 200 recording cylinders via a column-type access mechanism with 20 vertically aligned read/write heads, one read/write head per disk surface. Each cylinder position provides access to 102,400 bytes of storage.

Each drive accommodates one IBM 2316 disk pack (or equivalent) which contains 11 platters and 20 recording surfaces. The track-to-track head positioning time is 30 milliseconds with an average head positioning time of 74 milliseconds. The maximum positioning time is 156 milliseconds. The average transfer rate is 312,000 bytes per second and the rotational speed of the disk pack is 3600 revolutions per minute.

Model 6051 Magnetic Tape Drive

The Model 6051 Tape Drive has been one of ABC's standard workhorse models for many years. Although of lower data density and slower speed than more recent models, the 6051's low cost and high reliability account for its continued popularity in many installations. Since the Model 6051 uses industry standard 9 track 1/2 inch magnetic tape, it utilizes the same physical medium as all other ABC tape drives, thus requiring an inventory stock of only one tape type.

The Model 6051 records data at a density of 800 bytes per inch on standard 2400 foot length tape reels with 1/2

inch interblock gaps and using NRZ encoding. The tape speed is 37.5 inches per second and the average data transfer rate is 30,000 bytes per second.

Model 6110 Magnetic Tape Drive

The Model 6110 is ABC's standard high-speed 9-track tape drive. The high transfer rate of 120,000 bytes per second combined with the reliability of 800 bytes per inch data density makes the 6110 an effective tape storage drive. ABC's patented Quicloc mechanism speeds tape mounting. The 6110 has a tape speed of 150 inches per second and an interblock gap size of 1/2 inch.

Model 7001 Magnetic Tape Drive

The Model 7001 is ABC's top of the line high-speed 9-track tape drive. Although actual effective data storage capacity depends on the block size used, due to the need of a 0.4 inch interblock gap the Model 7001 allows a storage capacity of 40 megabytes. Unlike other ABC tape drives, the data density of the 7001 is 1600 bytes per inch. The tape speed is 150 inches per second and the average data transfer rate is 240,000 bytes per second. ABC's patented Quicloc mechanism speeds tape mounting. The Model 7001 has proven itself as a highly reliable tape drive in many installations.

Model 2920 Card Reader

The Model 2920 Card Reader reads industry standard 80 column cards at a rate of 1000 cards per minute. The input card bin holds up to 5000 cards.

Model 1200 Line Printer

The standard ABC chain printer, the Model 1200 prints on standard 11 by 14 inch fanfold paper at a rate of 1000 132-character lines each minute. The Model 1200 has an excellent performance record.

Model 1250 High-speed Printer

The belt-driven Model 1250 High-speed Line Printer is a quality line printer which prints 2000 132-character lines each minute. The Model 1250 is ABC's best line printer.

Model 1400 Laser Printer

The Model 1400 is a state of the art laser printer which prints 20,000 lines per minute. Variable pitch controls allow up to 187 characters on a single printed line. The Model 1400 prints on 20 pound roll paper and features variable size page cutting and optional three-hole punching, all of which are controlled by a microprocessor which is the heart of the Model 1400's operation.

Software Specifications

There are two different operating systems that can be run on the ABC BITBUCKET computer. These operating systems can handle either a single-cpu configuration or a multiple-cpu configuration.

Batch Uniprocessor Multi-Programming System (BUMPS)

BUMPS is ABC's multiprogramming operating system intended for use in a batch environment with a single processor. BUMPS comes as standard equipment with the BITBUCKET computer. This operating system is described in more detail in the following sections.

System Jobs

The BUMPS operating system consists of a nucleus which is core resident, requires 96K words, and includes such tasks as memory loading, input/output control, and processor scheduling. In addition there are three system programs which definitely have impact on the flow of user programs. The job scheduler, although core resident in the nucleus, must go through the execute queue like any other job. The input spooler and the output spooler are not normally core resident and must acquire resources in the hold queue like any user job. The operation of these programs and the resources they require is described in greater detail below.

Job Flow

A typical user program gets into the system through the input spooler (Figure 1). After the job is spooled, it is placed into the hold queue to await resources. The job scheduler examines the jobs in the hold queue and, according to job priority and the job's arrival time into the hold queue, allocates memory partitions and allocatable input/output devices to each job. When the job is allocated all of its required resources, it proceeds to the execute queue.. As the cpu becomes free, it takes the highest ranking job in the execute queue and performs a cpu burst.

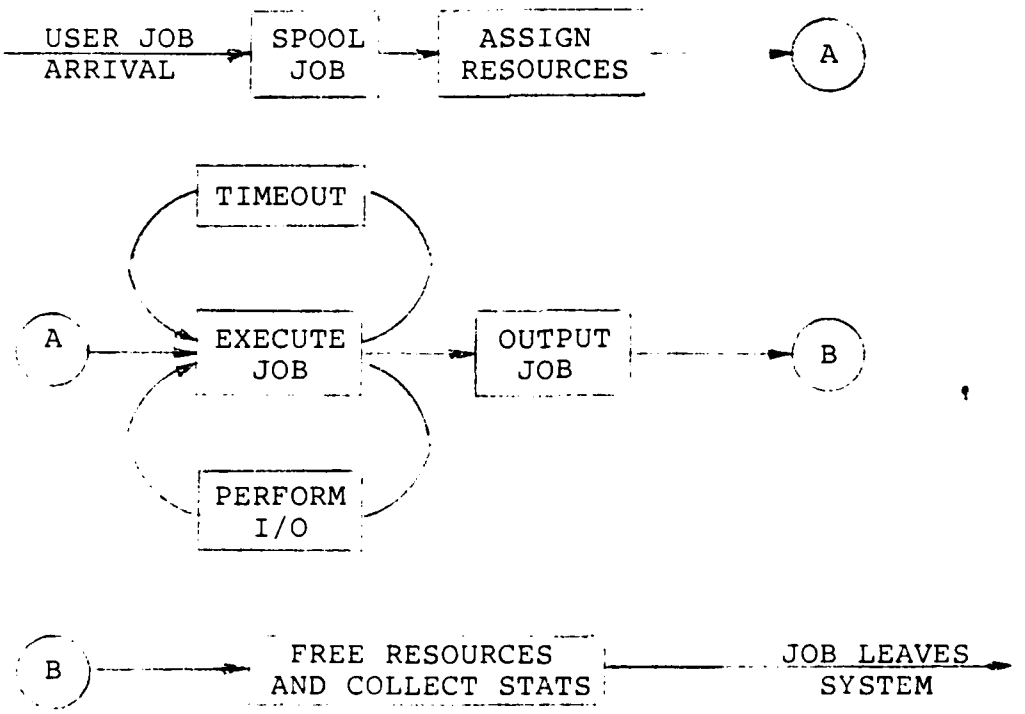


Figure 1. Typical Job Flow

Job ranking in the execute queue is done by job type. The job scheduler gets the highest rank, followed by the software monitor start-up job, output spooler, input spooler, and user jobs, respectively. The cpu burst continues until one of four things happen: an I/O is issued for an allocatable I/O device, an I/O is requested issued for an unallocatable device, a timeout occurs, or the job finishes. If an allocatable device I/O occurs, then the job gets placed in the smallest channel (IOM) queue which is connected to the requested I/O device. The I/O is performed, the channel is freed, and the job is placed back into the execute queue. If an unallocatable device I/O occurs, the job gets placed into the proper device queue. Once the device has been acquired, the job is placed in the smallest channel queue which is connected to the device. The I/O is then performed. After completion of the I/O, the channel and the device are freed and the job is placed back into the execute queue. If a timeout occurs, then the job is placed back into the execute queue to wait for another time slice. If the job is completed, then the job is placed into the output queue and it waits for the output spooler to print the job. Also upon job completion, the memory partition used and the allocatable devices assigned to the job are released back to the operating system.

Input Spooler

The input spooler takes the user's programs from the card reader and spools them onto the disk. When a new job arrives at the input queue, the operating system checks to see if the input spooler is already in memory. If not, the spooler is placed in the hold queue where it must compete for resources and cpu time like any other job. The input spooler requires a memory partition of 4K for execution. When the input spooler acquires the cpu, it first schedules an I/O to read the 80-byte cards into a 1K system buffer. After the buffer is loaded, the spooler is placed back into the execute queue. The next time the spooler gets the cpu, the buffer is spooled to the disk. This two-step process continues until the entire job is spooled. At that time, the operating system checks to see if there are any more jobs in the input queue. The input spooler continues until all the jobs in the input queue are spooled. The input spooler is then released from the system and the memory partition is returned to the operating system. The input spooler requires one millisecond of cpu time each time is acquires the cpu.

Output Spooler

The output spooler takes the job's output file which is stored on disk and prints it. Whenever a job arrives at the output queue, the operating system checks to see if the output spooler is loaded. If necessary, the output spooler

will be placed into the hold queue where it will wait until it can acquire a partition of at least 4K words. When the output spooler executes it will perform an I/O which will load a 1K system buffer from the disk file. The spooler will be placed back into the execute queue so that it can print the buffer in the form of 132-byte lines. This process continues until the entire job has been printed. Like the input spooler, the output spooler continues to operate until its associated queue is empty. In order to set up a transfer, the output spooler consumes one millisecond of cpu time each time it acquires the cpu.

Job Scheduler

The job scheduler allocates memory partitions and allocatable I/O devices (tapes) to jobs in the hold queue. The job scheduler is loaded into the execute queue (if it is not already there) each time a new job arrives in the hold queue or when resources are freed at the termination of any executing job. The job scheduler is always core resident as part of the operating system and therefore does not count against the multiprogramming level. When the job scheduler obtains the cpu, it looks at every job in the hold queue to see if it can assign resources. Resources are assigned to jobs according to a priority system. The job scheduler ranks the jobs according to each job's priority attribute. This attribute is part of the job's input parameters. If there

is a tie, the jobs are ranked according to hold queue arrival time. After ranking, the job scheduler checks to see if a memory partition which can handle the job is free. If so, the scheduler then checks to see if there are enough allocatable I/O devices available. When both conditions are met, the job is assigned resources and moved from the hold queue to the execute queue.

Static Partition Memory Management

With this memory manager, the user may specify up to 20 partitions of any size from 1K to 1000K words. Partitions are defined in terms of 1K word increments. The memory manager uses a "first fit" algorithm. When the job scheduler executes, it starts with the first free memory partition, checking to see if the job will fit. If necessary, the memory manager will check all free memory partitions.

Process Scheduler

The process scheduler selects jobs from the execute queue and forwards them to the cpu for a cpu burst. The process scheduler uses a priority round robin scheme. Jobs in the execute queue are ordered first by their job type and second by their arrival time to the queue. The job scheduler has the highest priority, followed by the software monitor start-up job, output spooler, input spooler, and user jobs, respectively. When the cpu becomes free, the process scheduler takes the highest-priority, longest-waiting job from the

execute queue. The job will execute until its time slice is used up, an I/O is issued, or the job terminates. If a timeout occurs, the job will be placed last in the execute queue. If an I/O occurs before the timeout, the job will perform its I/O and then return to the end of the execute queue. The process scheduler is part of the operating system nucleus and consumes no visible resources.

I/O System

To facilitate I/O data flow, all data transfers are done in fixed 1K blocks. The high-speed multiplexed ABC IOMs allow concurrent block transfers to or from several devices if the sum of the devices' effective transfer rates is within the transfer rate capability of the IOM itself. Each I/O request results in the transfer of a block of data from the specified devices, even if only a small portion of the block is requested. Inorder to make such a transfer, both the device and the IOM must be available. When an I/O is issued, the job first acquires the appropriate I/O device; if the device is busy, the job waits in the device queue. The device queues use a "first-come, first-served" algorithm. After the job acquires the device, it tries to allocate an IOM which is connected to the device. If all the IOMs are running at capacity, then the job will wait in the smallest of the channel queues. Only after the I/O device and the channel are free is the data transfer started. Where possible,

the I/O scheduler tries to distribute usage across the system of disks and drums.

Each job may have four types of I/O. First it must read all of the spooled cards from the disk. It must write all printer lines to the disk file (for later transfer by the output spooler). In addition, each job may require a number of disk/drum I/Os and a number of tape I/Os.

Batch Multiprocessor Multi-Programming System (BMMPS)

The BMMPS operating system is exactly like the BUMPS operating system, except that it supports multiple processors. BMMPS can have a maximum of 99 cpus connected at one time. There is still, however, only one execute queue from which all the cpus acquire jobs to process. Thus, a job may at one time or another be executed on all of the system cpus during a single run. All of the cpus in the system appear identical to the operating system and each can process any job.

Accounting Data

The ABC BITBUCKET computer records accounting data on every user job that goes through the system. The accounting file contains one record per job. The contents and the file format are listed in Table I.

TABLE I

ACTLOG Record Format

FIELD	VARIABLE	FORMAT
1	Arrival Time	1X,F15.4,1X
2	Job Nme	F10.0,1X
3	CPU Time	F5.0.1X
4	Memory	F5.0,1X
5	Priority	F5.0,1X
6	Allocable Devices	F5.0,1X
7	Cards	F6.0,1X
8	Lines of Print	F6.0,1X
9	Disk blocks	F6.0,1X
10	Alloc'able Dev. Blocks	F6.0,1X
11	Job Type (Note 1)	F2.0,1X
12	CPU Time Used	F10.3,1X
13	I/O Time Used	F10.3,1X
14	Memory Size Used	F10.3,1X
15	Departure Time	F15.4

NOTE 1: ACTLOG records only user jobs, so job type will always be 1.0.

Software Monitor

The software monitor is like any other job in the ABC BITBUCKET computer system. It is entered into the system and must compete for computer resources. The monitor is loaded into the hold queue at the user-specified starting time. It stays in the hold queue until a partition of at least 4K is available. It then moves from the hold queue to the execute queue where it stays until it can acquire the cpu. At that point, the monitor begins the tracing of jobs through up to five user-specified queues. The software

monitor then releases the cpu back to the operating system. Once the trace has started, the monitor will record data about every job that enters one of the five monitored queues. The queue name, the time spent in the queue, where the job came from and where the job is going is recorded every time a job leaves a queue. When the software monitor is monitoring queues and recording data, it puts an additional burden on the system which causes the computer to run at a maximum efficiency of 95%. When the scheduled stopping time of the monitor occurs, the memory partition is freed and the cpu returns to 100% maximum operating capability. The software monitor writes its data to a file called SWMON followed by the student's team number. This file is available for post-processing. The contents and record format of the file appear in Table II.

TABLE II
SWMON Record Format

FIELD	VARIABLE	FORMAT	COMMENTS
1	Queue Name	1X,A15,1X	See Note 1
2	Time in Queue	F8.3,1X	In seconds
3	Came from?	A15,1X	See Note 1
4	Going to?	A15	See Note 1

NOTE 1: See Table III for possible contents of these variables.

TABLE III
Possible SWMON Names

Variable Nme	Comments
JOB ARRIVAL	
INPUT QUEUE	
HOLD QUEUE	
EXEC QUEUE	
OUTPUT QUEUE	
ARVL SPOOL	Input Spooler arrival in system
ARVL OSPOOL	Output Spooler arrival in system
ARVL S/WMON	S/W Monitor arrival in system
ARVL JSCHED	Job Scheduler arrival in exec. queue
JOB FINISHED	
TAPE 1 QUEUE	
TAPE 2 QUEUE	
TAPE 3 QUEUE	
TAPE 4 QUEUE	
TAPE 5 QUEUE	
TAPE 6 QUEUE	
TAPE 7 QUEUE	
TAPE 8 QUEUE	
TAPE 9 QUEUE	
TAPE 10 QUEUE	
DISK 1 QUEUE	
DISK 2 QUEUE	
DISK 3 QUEUE	
DISK 4 QUEUE	
DISK 5 QUEUE	
DISK 6 QUEUE	
DISK 7 QUEUE	
DISK 8 QUEUE	
DISK 9 QUEUE	
DISK 10 QUEUE	
CHAN 1 QUEUE	IOM 1 queue
CHAN 2 QUEUE	
CHAN 3 QUEUE	
CHAN 4 QUEUE	
CHAN 5 QUEUE	
CPU	
GENERAL CHAN	Used only in "Going to?" because channel # not yet determined
SPOOL QUEUE	Input Spooler queue
OSPOOL QUEUE	Output Spooler queue

Hardware Monitor

Unlike the software monitor, the hardware monitor does not use up the BITBUCKET computer resources. The hardware monitor has its own timers, counters, and data recording devices. It is an event-driven monitor which can be connected to any I/O device, IOM, or cpu. The hardware monitor has two timer probes and three counter probes. These probes, along with the starting time, stopping time, and the sample rate, are specified in the configuration file. One limitation to the hardware monitor is that the monitor treats multiple cpus as one. The timer or counter connected to the cpus will be pulsed every time any cpu is activated. The output of the hardware monitor is written to a data file at the end of each interval specified by the sample rate. The output file is called HWMON and the record contents and format appear in Table IV.

TABLE IV

HWMON Record Format

Field	Variable	Format
1	Time of Recording	1X,F15.4,1X
2	Timer #1	F8.3,1X
3	Timer #2	F8.3,1X
4	Counter #1	I5,1X
5	Counter #2	I5,1X
6	Counter #3	I5

III. Modifying the ABC Computer Configuration

The ABC BITBUCKET computer can be modified to any rational configuration by using the menu-driven program cpesim. The program stores the configuration into the database for later extraction by the simulation.

Program cpesim

Program cpesim was designed to be virtually independent of this manual. However, some written documentation is always necessary in the event of unanticipated confusion. To execute cpesim, the student needs the location of the program from his instructor. The student may or may not wish to change his working directory to the directory containing cpesim; in either event, it is unnecessary and will make no difference in program execution. To begin, the student need only enter the word cpesim into the terminal (followed, of course, by a carriage return) and the program's "on-line help" mechanism will take over. The first thing the student will be queried for is the current quarter. Valid choices are fa (fall), wi (winter), sp (spring), or su (summer) followed by the last two digits of the current year. An example appears with the program's prompt. The student will then be asked for his team number. This number will be assigned by the student's instructor. Then the student will be asked to enter the password that his instructor has assigned for

his team. The program will take a few moments to verify this password against the one stored in the database for his team number. An example of this first interchange appears below.

```
What is the current quarter?
(Example: wi85)
fa84

What is your team number?
2

Please enter the password for team 2
dentist

Verifying password...
```

If the password as entered is not valid, the student will receive the message

"You are not authorized access to that information."

and the program will terminate.

If the student's password matches the stored password for that team, he will be offered the following menu:

```
What would you like to do?

     1) Display a configuration
     2) Change a configuration
     3) Process software monitor data
     4) Exit the program

Please enter choice:
```

If either of the first two menu items is picked, cpesim will ask the student for the configuration number that he wants. Configurations are stored by week. If the initial configuration is to be retrieved, the instructor will have to give the students the week number under which it is stored; very likely it will either be zero or one. If a configuration has already been retrieved in this interactive session, cpesim will remind the student what number configuration has already been loaded.

```
Which week's configuration do you want?
(Current week is #1)
```

If a new configuration is to be retrieved, cpesim will then display "Retrieving configuration..."

Displaying a Configuration

Once the desired configuration is loaded (some configuration will always be loaded before cpesim tries to display it), cpesim will prompt "Enter 'go' for XXX" where XXX is some portion of the configuration (e.g. CPUs, IOMs, partitions, etc.). This is done to keep large configurations from scrolling off the screen before the student has a chance to examine them. The student need not enter 'go' specifically to see the next portion of the configuration. Hitting any key followed by a carriage return will perform the same function, but 'go' is a short word and adequately describes the intention, so it was selected as the mnemonic response.

Changing a Configuration

Changing a configuration is a very simple matter with cpesim. After the configuration to be changed has been loaded, the following menu will appear:

What would you like to do?

1) Change CPU
2) Change memory module
3) Change memory partition
4) Change peripheral device
5) Change monitor usage
6) Change IOM
7) Change peripheral - IOM connection
8) Redefine all memory partitions
9) Save all changes for simulation
10) Abort all changes

Please enter choice:

Selection of any but the last three options will put the student at another menu which will generally resemble this one:

What would you like to do?

1) Add a CPU
2) Remove a CPU
3) Replace a CPU
4) Change the time slice
5) Display CPUs
6) Return to previous menu and save CPU changes

Please enter choice:

These lowest menus generally put the student into the routines where actual changes to the configuration are entered. Any number of changes, even to a changed item, may be made. The last menu option refers to saving changes. This "save"

means to save changes in the "accumulate" sense, rather than in the "write to database" sense. The option to write the changes to the database appears in the main change configuration menu.

All the menus have been designed to be as informative as possible. In general, the student need not fear the lowest level change menus as any change made in error can itself be subsequently changed. If the student finds that he has made so many errors that it would be easier to start over rather than try to change the errors, it would be best to pick the "Abort all changes" option in the main change menu. All changes that the student has made will be discarded and the configuration he began with will be intact. It is better to pick this option than to "break" out of the program as picking the abort option means avoiding another configuration retrieval which, during peak system load, may take several minutes. Configuration changes will not be saved to the database until the "Save all changes for simulation" option is picked. The student is cautioned not to pick this option until he is sure that all the desired configuration changes have been made. Once changes are saved to the database, they cannot be altered; they can only be the basis for a new week's configuration. The student has no power to change this situation. Only under the most extreme of circumstances might the instructor be persuaded to save a student from this

dilemma; it is a most painstaking and time-consuming procedure which only the instructor has the ability (and access permissions) to perform.

Processing the Software Monitor Data

When the student picks this option, he should have the file containing his software monitor data (SWMONX, where X is the student's team number) in the working directory. The cpesim program may appear to do nothing more than pause a few moments and then re-display the same menu when this option is picked. What has, in fact, happened is that the cpesim program has called another program to process the file and it is now doing so in background mode. This program may take some time to complete and it is not anticipated that the student wishes to stare at a frozen terminal screen while it is doing so. Thus, the job is spawned as a background job and the student may later look for results in a file called SWMONX.dat where X is the student's team number.

The SWMONX.dat file will contain entries, one per job that entered the BITBUCKET computer, which summarize the time spent by the jobs in the various queues. Also, each queue will have a histogram which describes the distribution over the population of jobs of time spent in that queue. The student may find this information useful in his analysis.

Addendum

The cpesim program was designed to be as homogeneous as possible. The menus were purposefully designed with the same construct to breed comfort through familiarity. This strategem may have been carried too far, however. At first glance, many of the program's menus look alike, so the student is cautioned to glance a second time to be sure that he does, indeed, have the proper menu before him. This precaution will help prevent errors and unwanted configuration changes.

Appendix E

CPESIM II

Instructor Manual

Contents

List of Tables

CPESIM II
Instructor Manual

I. Introduction

CPESIM II consists of a major computer system simulation and its operating emvironment. It was designed as a laboratory aid to allow students to apply computer performance evaluation (CPE) tools and techniques to a hands-on computer system. The use of an actual operating computer system by students for such CPE studies is impractical for several reasons. First, the overhead required and the disruption caused by some measurement tools is prohibitive to use in an actual computer installation. Second, if measurement data is available, it may not be academically useful; the system may be operating as it should and therefore providing no problem-solving opportunities, or it may suffer under several interacting problems which, although realistic, cannot practically be resolved by a student as a one-quarter course project. Finally, if the data does permit the student an analysis and solution proposal, an operating computer installation is unlikely to allow solution implementation or to be able to choose from among conflicting solutions proposed by several students. CPESIM II solves these constraints by providing a simulated environment in which the student can gather whatever data is desired, analyze it, recommend and implement a feasible solution, and finally

verify the effectiveness of the modified system.

CPESIM II consists of two parts. The heart of CPESIM II is a computer simulation (written in SLAM) of a large-scale computer system which executes on a host machine -- the VAX system. Because it is a simulation, there are many simplifications and limitations which restrict its representation of reality. The second part of CPESIM II is a simulated operating environment for the computer system simulation. This includes a computer-generated workload, a written scenario, budgetary constraints (if desired by the instructor), and system data in a form that can be manipulated and analyzed by the student, who is playing the part of a CPE analyst and/or a data processing manager.

The written scenario provides the student with operating details about the particular computer installation he is to investigate. The instructor configures the hardware, operating system, and workload to illustrate a particular problem or situation. Measurement data is then provided as requested to the student for analysis and decision-making on a periodic (e.g. weekly) basis. In order to force the student analyst to make tradeoffs (a real-world constraint), only a subset of the available data can be accessed during a given period.

CPESIM II Output to Students

Many types of information are available to the student to aid in the analysis. Some of these are free and

automatic; some must be specifically requested. There is a cost associated with many, while some are mutually exclusive. The student must choose what data he wants and how he wants to use it. This section briefly describes the outputs available and directs the student to further information where it is available.

Manufacturer's data -- Literature of varying value which describes hardware and software features is available in printed form.

Installation documentation -- In-house documentation of the local configuration, parameter values, modifications, problems, etc. may be available in printed form. These are specific to a given problem and will be provided, if available, as a separate handout.

Accounting system data -- The computer's accounting system files are available at no cost to the student in a file on the VAX named ACTLOG followed by the student's team number. Such data is in raw form and the format is defined in the manufacturer's literature for the appropriate operating system.

Hardware monitor -- A hardware monitor exists for the simulated computer, but may have to be purchased or leased. Particular probe points, sampling periods, and sampling frequency must be specified by the student. As with a real hardware monitor, its use has no effect on system performance. The output data is

available in a file on the VAX named HWMON followed by the student's team number.

Software monitor -- A software monitor is available for the simulated computer, but, like the hardware monitor, may have to be purchased or leased. As with real software monitors, this monitor requires memory, runtime, and system overhead to operate. Output data is available in a file on the VAX named SWMON followed by the student's team number.

Student Inputs to CPESIM II

One of the advantages of a simulation is that students can easily make radical (or not-so-radical) changes to the system. Thus, there are a number of inputs that the student may (or must) provide to CPESIM II as well as some that he cannot. This section briefly describes the CPESIM II inputs available to the student.

Workload -- The student has no control over the workload at all. The instructor controls the workload and may choose to vary it as often as he desires or in response to situational developments (e.g. in response to a student-initiated "user education program").

Interview requests -- Student analysts may submit written questions to "installation personnel". The questions, however, may or may not be answered.

Monitor purchase -- Student analysts may submit purchase orders for the purchase or lease of any available hardware or software monitors.

Monitor input parameters -- If a monitor is being used, students must provide to the simulation a starting and stopping time for monitor operation. The software monitor, in addition, needs to know which queues are to be monitored, while the hardware monitor needs to know where the probes are to be connected and what the sample rate will be. These parameters are input while making any other changes to the system configuration prior to simulation run time.

Reconfiguration requests -- Students may direct reconfiguration of the existing hardware and operating system parameters. Any degradation of system performance, however, will be frowned upon by installation management.

Purchase of system options -- Additional hardware or operating system modules may be purchased and installed by submitting an appropriate purchase order. Information on current cost and availability will be provided with each particular project.

Personnel hiring -- Requests for hiring additional personnel (e.g. for an additional shift) or for overtime authorization may be submitted.

Student Input for Grading

Grading structures are, of course, the prerogative of the individual instructor. This section describes a potential grading structure which might be used in conjunction with CPESIM II to grade a CPE project.

Final Report: The final report represents the single most important input to the student's final grade. The report should be typewritten and should include drawings, tables, and graphs of a professional quality. The report should consist of two parts; the analyst's report and the student's critique. The analyst's report should include a clear statement of the perceived problem, a systems analysis and description, stated hypothesis of the specific problem(s), analysis done to confirm or deny the hypothesis, recommended solutions(s), and verification that the implemented solution(s) solved the problem(s). Also included should be a cost analysis and recommendations for the future. The student critique should include an evaluation of the CPESIM II system as a learning tool along with a discussion of problems encountered and recommended changes.

Interim reports: Various interim reports may be required throughout the quarter, depending upon the scenario. These should be of quality similar to that of the final report and will count toward the project grade.

Software tools: Some specific software tools (e.g. data reduction packages) may be required during the quarter. These will count toward the project grade. For any tools

that the student may develop, the source code should be submitted as an appendix or addendum to the final report. These programs will influence the final grade.

Oral presentation: Each team will be required to make a final oral presentation to the class and possibly an interim oral status briefing as well. Although the central purpose of the final presentation is to share each team's analysis and findings with the rest of the class, the presentation is a graded part of the project and should therefore be done in a professional manner.

CPESIM II Interface Structure

The instructor's interface to CPESIM II, program cpectl, has five main functions: generation of the simulation workload, generation of the hardware catalog, generation of the initial system configuration, listing the configuration histories, and running the simulation. These functions are utilized via a system of helpful menus in cpectl and are, for the most part, self documenting. As a additional aid to their utilization, however, they are covered in some detail where necessary in the following sections.

II. Workload Generation

Introduction

Generation of the simulation workload is actually accomplished by a FORTRAN program called jobgen. The program, called by the interface program cpectl, allows the instructor to create the workload with from one to ten jobstreams, each of which can be completely different from or virtually identical to another. The instructor must select, for each jobstream, a frequency distribution concerning various job parameters. The details for this are covered in the two sections which follow.

Frequency Distributions

There are eleven different frequency distributions in cpectl from which to select. The instructor may create a rather large variety of workloads by selecting one distribution from the eleven to describe each job parameter. Possible frequency distributions include exponential, uniform, Weibull, triangular, normal, lognormal, Erlang, gamma, beta, and Poisson distributions. These distributions were selected as being the most often used in computer simulation. If these are insufficient, however, the instructor may describe his own step distribution to cpectl and it will be used to describe whichever job parameter he selects.

Job Parameters

Each of several parameters must be described by one of the aforementioned distributions in order for the workload to be generated. These parameters include job interarrival time, CPU time required for job completion, memory required (in 1K blocks) by a job, a relative job priority, the number of allocatable devices required by a job, the number of input cards in a job, the number of lines a job may output, the number of system disk blocks needed, and the number of allocated device blocks a job might need. Program cpectl will go through this list of parameters for each job stream and query the instructor for a distribution to describe it. The program will also offer the instructor the chance to provide his own random number seed; if one is not provided, the seed has a default value. The program will then actually generate a simulation workload and put it in a formatted file with whatever name the instructor has input to the program. The procedure for all this is better explained through actual execution rather than by this description. The instructor is referred to the program itself for the best instruction.

III. Hardware Catalog Generation

Generation of the hardware catalog for the CPESIM II system is a very simple process. The instructor is placed into a query loop which asks for a model number, a data rate (or size or speed factor), a name for the item (up to 20 characters), a device type, and an associated cost. To exit the loop, the instructor enters a model number of zero. The instructor will then be given the opportunity to change, delete, or add to anything he has entered into the catalog. When the instructor is satisfied with what he has entered, he picks a menu option which causes the data to be stored into the database.

This same program can be used to make later changes to the catalog. This is a menu-driven operation and is best explained through actual use of the program. The menus allow the instructor to add, delete, or change any item in the catalog. This operation, however, is a "real-time" operation with the database. That is to say, unlike the student program which makes changes to the configuration first and then stores the new configuration, this operation stores each change into the database as it is entered. If the system user load is high, this can be a time-consuming affair.

IV. The Initial Configuration

The initial BITBUCKET computer configuration is created by virtually the same set of modules that the students use to change the configuration. The only significant differences are that the instructor's version of the modules do not try to retrieve a configuration to make changes to and the program offers the user the ability to specify what week the configuration will be stored under. For more detailed instruction on the use of these modules, the instructor is referred to the student manual. The query for which week the configuration will be stored under follows the selection of the menu item "Save the configuration to the database".

V. Listing the Configuration Histories

Like the creation of the initial configuration, the listing of student team configuration histories is accomplished by virtually the same modules that the students use. There is a greater difference in these instructor modules, however. The instructor may display any one configuration on the terminal, display all the configurations for one team, display all the configurations forall teams, or perform these same functions writing the output to a file. Students may list only a single configuration at a time and only on the screen. Also, they can only list their own configurations, while the instructor can list the configuration of any team. The selection of these instructor options is achieved via a menu.

This list option is intended to allow the instructor to follow the progress or evaluate the achievement of his student teams. He may use it to spot check any given configuration as well. Students should at least summarize their configuration evolutions in their final reports, but this option "offers the story" at a glance. It is suggested that the instructor utilize this option in conjuction with his evaluation of the student's final results. With a complete historical record before him, he can see what the student has done and where the student has made any errors.

VI. Running the Simulation

A minimum of input is required from the instructor to allow the execution of this option. The instructor may start the simulation for all student teams or only one student team. He may also, at his discretion, abort any simulation just prior to its execution.

When the "run simulation" option is selected, many things happen which are invisible to the instructor. He is first prompted for the week number of the simulation he is running and the name of the workload file that he will use. He will be asked if he wishes to change the default simulation run length. He then selects, from a menu, whether he wants to run the simulation for all teams or for one team. Selecting the latter option is useful if, for some reason, a student team failed to store a configuration for the current week prior to simulation run time. (If this situation occurs, the program will inform the instructor that a particular team's data is missing and continue processing the next team's data; no program abort will occur.) This option allows the simulation to occur at another time (at the discretion of the instructor, of course) when the configuration has finally been stored and without having to unnecessarily rerun a simulation for the other teams. If the instructor selects a one-team simulation run, he will be asked for the number of the team for whom the simulation is being run. If he wishes to start a simulation for all teams, he

will be asked how many teams there are.

Another rationale for a one-team simulation is that perhaps one team has instituted a user-education program which impacts the system workload. For that team, then it would be necessary to run the simulation with a different workload, one more akin to the results of their influence. The workload, of course, need not necessarily reflect a positive effect due to their influence. It may, in fact, be the case that the "simulated users" rebel against any changes made by the students and actually cause the conditions existent to worsen! This, of course, would all happen at instructor discretion.

The program begins to do several things when all the parameters have been entered. First the SLAM network code is copied into a simulation input file. The program then extracts a configurations from the database and copies it into the simulation input file. Next, the workload file is copied. The file is closed and the instructor is offered the opportunity to abort the simulation. This is done to allow the instructor to create the input file without actually running the simulation. This file may then be sent to a simulation other than the standard CPESIM II simulation, perhaps a modified version. Simulation results appear in the files which are documented in the student manual.

Setting up the CPESIM II System

At some point in time, it will be necessary to take the CPESIM II source code and make it executable. This section was written to make that task a simple one. Table I contains a list of all the needed files and indicates what they are.

The Interface. Four programs make up the CPESIM II interface: one FORTRAN program (jobgen) and three C programs (cpesim, cpectl, and procmon). Program jobgen should be compiled as follows:

```
F77 jobgen.f -o jobgen -limsl
```

Two of the C programs, cpesim and cpectl, contain embedded EQUEL code which will have to be precompiled. This is accomplished with the following:

```
equel cpesim.q
equel cpectl.q
```

EQUEL will produce a X.c file (where X was the name of the file with the embedded EQUEL code minus the ".q") for each of these. Now all three C programs can be compiled with

```
cc -w cpesim.c -o cpesim -lq
cc -w cpectl.c -o cpectl -lq
cc procmon.c -o procmon
```

These commands produce the files cpesim, cpectl, and procmon respectively. If the source files exist on a backup medium (removable disk or tape), the X.q and the X.c files can be deleted.

The cpesim and procmon files should be moved to a directory which has global read, write, and execute permissions. These are the student programs and the directory containing them becomes known as the student work area. The directory is likely to become somewhat littered with files, so it is a good idea to start it out empty except for the two interface programs. The other programs, cpectl and jobgen, should be placed where only the instructor has permission to execute them.

The Simulation. Two files comprise the CPESIM II simulation: sim.f and sims. Only one of these files needs to be compiled. Issue the following command to UNIX:

```
F77 sim.f -o simf -limsl
```

to create the simulation object code. Once the program is compiled, if the source files are on a backup medium, delete the sim.f file. Move files simes and simf to the directory containing cpectl.

Using the System. As the instructor runs each simulation, several data files will be created for the students. He has two options: he can allow general read permissions on the directory containing cpectl, jobgen, simf, and sims

without giving any general permissions to the aforementioned programs, or he can copy all the students' data files to the student work area. The former option involves less work. With global read permissions (only) on the directory, the instructor can set general read permissions on the data files (if they don't already have them by default) and the students can copy their data files to their own areas. Advise students to do this because the existing files will be overwritten during the next simulation run.

TABLE I.

CPESIM II Files

Filename	Contents
cpesim.q	Source code for the student interface. Contains embedded EQUEL code.
cpectl.q	Source code for the instructor interface. Contains embedded EQUEL code.
jobgen.f	Source code for the workload generator. (FORTRAN)
procmon.c	Source code for the software monitor data post processor.
sims	Network simulation code. Do not try to compile!
sim.f	FORTRAN simulation code.

VII. Summary

The instructor's interface to CPESIM II offers a variety of options and functions. These functions include generation of the BITBUCKET system workload, generation of the catalog of hardware available to the installation, generation of the initial BITBUCKET configuration, displaying the histories of the final (or current) configurations developed by each student team, and initiating the simulation run. The operation of each of these functions is facilitated by helpful and informative menus which were designed to operate independently of written documentation.

Appendix F

Development of a User Support
Package for CPESIM II
(A Computer Simulation for CPE Use)

Publishable Article

Development of a User Support
Package for CPESIM II
(A Computer Simulation for CPE Use)

Recently, more and more computer installations are turning to computer performance evaluation (CPE) as a first means to providing more effective computer support. In order to learn CPE techniques, students may best profit from a computer simulation. Such a simulation was developed in SLAM in 1983 by Capt. David Owen as an upgrade of a similar simulation developed in 1979 in SIMSCRIPT II by Capt. Paul Lewis. However, Capt. Owen's simulation, CPESIM II, lacked the user friendly interface which should characterize a learning tool. This thesis effort was an attempt to rectify this situation while at the same time meeting some new requirements from CPE and Queueing Theory classes at AFIT.

The new CPESIM II system was required to allow students to easily display or modify an initial computer configuration which was part of a case study in performance evaluation. The students were to be allowed to make any number of changes over the course of several weeks in order to resolve the performance problem being illustrated and restore effective computer support. The students were to be able to add, remove or replace any hardware they deemed necessary to achieve their goal. Data for analysis of the system was to be obtained through hardware and software monitors which are available to sample a student-defined subset of all the computer's

performance data and from the accounting log of the computer. The interface was also to be capable of doing a data reduction and histogram analysis of the output software monitor data. All configuration changes were to be stored in a database in order to provide the instructor with a stored history of the evolving configurations.

The new system was also to provide support to the instructor. He was to be able to easily create the initial computer configuration, a catalog of equipment (hardware) available for use, and a random workload of from one to ten jobstreams. The parameters of each job in the workload were each to be defined by one of eleven frequency distributions supplied to the instructor, one of which was to be an instructor-defined discrete probability distribution. The instructor was also to be able to display any or all configurations of any or all student teams and to be able to easily initiate the simulation run.

All software in the interface program was required to be portable: that is, to run on any UNIX system and be independent of the hardware, terminals included, that it was implemented on. This would allow the use of CPESIM II outside of the AFIT/EN environment and also allow the system to withstand any changes in the hardware it was implemented on.

The finished interface was implemented in the C language and heavily depends on the use of the Ingres relational DBMS. Starting with the initial configuration, all student-changed configurations are stored in an Ingres database to be

displayed by the students and extracted for use at simulation runtime. Students are prevented from seeing the configurations of another student team through a team password construct. They are also prevented from utilizing any instructor capabilities by the fact that the interface is, in fact, basically two separate programs. The student program exists with global read and execute permissions while the instructor program exists with permissions allowing only the instructor to access it.

The student program allows the student to modify or display the initial configuration or any configuration modified by his student team. He may also do a data reduction and histogram analysis on the software monitor data after the completion of the simulation run. The program is driven via a set of hierarchically arranged menus which do actual data manipulation only at the lowest levels. It was felt that the menu structure is so commonplace as to be familiar to students and thereby shorten the learning curve for its use. Also, menus can be a very user friendly construct and can best filter out erroneous user entries into the system.

The instructor program, also menu driven, allows the instructor to create the initial configuration and display the students' configurations using virtually the same modules the students use to accomplish their tasks. The instructor, however, may direct his displays to a file for later printing or may elect to display on the screen; students may only display on the screen. The instructor may also create

as many workloads as he needs as defined by the requirements. He must do so before running the simulation which will use them, however. When the instructor chooses to start the simulation, the SLAM network code, the configuration (which is extracted from Ingres at this time), and the workload are all copied into one file to be used as simulation input. Capt. Owen's simulation required these to be separate files, but the new method reduces the plethora of files which would soon litter the system. The program also allows the instructor to easily create the equipment catalog which is also stored in the Ingres database with the configurations.

Upon completion, the system was programmer verified and then subjected to some scrutiny by 20 volunteers, who provided some valuable feedback which resulted in several cosmetic changes. The volunteers found the interface easy to use and understand, but had complaints about the speed of Ingres accesses when the UNIX user load was high. Validation of the CPESIM II system as a whole was not possible, partly because it was known that the simulation terminated abnormally after throughputting 50 jobs and partly because of difficulty getting the simulation reloaded onto the system owing to an intervening operating system change. After a "best guess" scrutinization of the system as a whole and especially the simulation input, the system was declared "probably valid", the best that could be done under the circumstances.

The interface achieved its objectives, but has room for improvement. It could benefit from the implementation

of a "help" system which would provide on-line aid to users. It could also benefit from a feature to track monetary costs of student changes and actions. Also, the programs did not have a terminal-independent clear screen function. Such a function is possible, but time constraints prevented its development. Currently, the clear screen function operates only for VT100 terminals. Also, workload parameters are not stored in Ingres, so they must all be entered each time a new workload is created. It would be less taxing on the instructor if the parameters were stored so that a few changes only would be necessary to create a new workload. At present, the hardware monitor data must be taken as is; no boolean operations on probe points are allowed. A change to effect this would improve the quality of the analysis data. Finally, the simulation itself requires some error corrections and streamlining. Since this was outside the scope of this effort, no work was done in this area. When these recommendations are effected, a truly superior learning tool will result.

VITA

Captain Daniel L. Petty was born on 14 January 1955 at Wright-Patterson Air Force Base, Ohio. He graduated from high school in Falls Church, Virginia, in 1973 and attended the University of Georgia from which he received the degree of Bachelor of Science in Computer Science in 1978. In January 1979 he entered Officer Training School and was commissioned in April 1979. He was assigned to the Air Force Data Services Center, Pentagon, Washington DC, where he served as Financial Systems Programmer/Analyst, Chief, Presidential Budget Computer Support, and Presidential Budget Computer Systems Analyst until entering the School of Engineering, Air Force Institute of Technology, in June 1983.

Permanent address: 8807 Gramercy Lane
Laurel, MD 20708

Bibliography

1. Bell, T. E., B. W. Boehm and R. A. Watson. Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort. A report prepared for United States Air Force Project Rand, November, 1972. (R-549-1-PR).

2. Lewis, Paul C. A Computer Performance Evaluation Education Tool. MS Thesis GCS/EE/79-8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, December 1979 (AD A080154).

3. Owen, David L. CPESIM II: A Computer System Simulation for Computer Performance Evaluation Use. MS Thesis GCS/EE/83D-16. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, December 1983.

4. Hartrum, Thomas C., Computer Performance Evaluation Instructor. Personal interview. Department of Electrical Engineering, School of Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Oh., 14 May 1984.

5. -----, Computer Performance Evaluation Instructor. Personal interview. Department of Electrical Engineering, School of Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Oh., 18 July 1984.

6. Seward, Walter, Queueing Theory Instructor. Personal interview. Department of Electrical Engineering, School of Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Oh., 18 July 1984.

7. Pritsker, A. Alan B. and Pegden, Claude Dennis. Introduction to Simulation and SLAM. Halsted Press, New York, NY, 1979.

8. Svoboda, Liba. Computer Performance Measurement and Evaluation Methods: Analysis and Applications. American Elsevier Publishing Company, New York, NY, 1976.

9. Ferrari, Domenico. Computer Systems Performance Evaluation. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

10. Kenighan, Brian W. And Ritchie, Dennis M. The C Programming Language. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

11. Plum, Thomas. Learning to Program in C. Plum Hall Inc., Cardiff, NJ, 1983.

12. Ageloff, Roy and Mojena, Richard. Applied FORTRAN 77 Featuring Structured Programming. Wadsworth Publishing Company, Belmont, CA, 1981.

13. Madnick, Stuart E. and Donavan, John J. Operating Systems. McGraw-Hill Book Company, New York, NY, 1974.

14. Becker, Richard A. and Chambers, John M. S: A Language and System for Data Analysis. Bell Laboratories, 1981.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Petty, Daniel Lester, Capt. USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1984 December; 15. PAGE COUNT: 150	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	CPE, Computer Performance Evaluation, Computer Simulation, User Friendly Interface
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: DEVELOPMENT OF A USER SUPPORT PACKAGE FOR CPESIM II (A COMPUTER SIMULATION FOR CPE USE)

Thesis Chairman: Dr. Thomas Hartrum

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Thomas Hartrum	22b. TELEPHONE NUMBER (Include Area Code): 513-225-2024	22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

In 1983 a SLAM computer simulation was developed to be used as an educational tool in CPE and Queueing Theory classes at AFIT. Lack of user friendliness and additional requirements necessitated the development of a user friendly interface, i.e. this thesis effort. The interface consists of two programs: one for the instructor and one for his students.

The students' interface allows them to modify and display the initial or subsequently modified computer configurations and to do a data reduction and histogram analysis on output software monitor data after a simulation run. Any student configuration changes are stored in an Ingres database.

The instructor's interface allows him to easily create the initial configuration, the catalog of available hardware, and the system's workload of from one to ten jobstreams. He can also display any or all configurations of any or all student teams on his terminal or direct that display to a file for later printing.

The interface as a whole is menu-driven, user friendly, and very portable; it operates on any UNIX system (which has Ingres and SLAM) regardless of the hardware (including terminals) that the operating system is implemented on. A student and instructor user's manual is provided.

END

FILMED

5-85

DTIC