TOOLS FOR RESEARCH IN COMPUTER WORKLOAD CHARACTERIZATION AND MODELING

by

Steven Louis Gaede

Memorandum No. UCB/ERL M79/58

1 September 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Tools for Research in Computer Workload Characterization and Modeling

Steven Louis Gaede

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720

and

Bell Telephone Laboratories
Naperville, Illinois 60540

ABSTRACT

Workload models are required in many Computer Performance Evaluation studies. Workload models must be *representative* of the workload that they are intended to model. Different definitions of "representativeness" bring about different workload modeling techniques. Each workload modeling technique has its advantages and disadvantages. A performance-oriented approach to the problem shows promise of overcoming its difficulties. Further work with this approach is needed, and tools are required to support further study. A Benchmark Driver, a Workload Analyzer, a Workload Extractor, and a Subworkload Analyzer are designed and built to satisfy the need for tools. The set of tools constructed for this purpose support specific experiments with a performance-oriented workload modeling methodology, and they support long-range research in workload characterization and modeling.

September 1, 1979

This research was supported in part by Bell Telephone Laboratories, and by the National Science Foundation under grants MCS78-24618 and MCS78-07291.

Tools for Research in Computer Workload Characterization and Modeling

Steven Louis Gaede

CONTENTS

1. INTRODUCTION 1

2. THE WORKLOAD MODELING PROBLEM 3
   2.1 The Resource Consumption Approach 4
   2.2 The Functional Approach 6
   2.3 The Performance Oriented Approach 7

3. TOOLS FOR WORKLOAD CHARACTERIZATION AND MODELING 11
   3.1 The Benchmark Driver 13
   3.2 The Workload Analyzer 17
   3.3 The Workload Extractor 18
   3.4 The Subworkload Analyzer 20

4. USE OF THE CHARACTERIZATION AND MODELING TOOLS 24
   4.1 A Workload Modeling Experiment 26

5. CONCLUSION 30

6. REFERENCES 33

APPENDICES
   A. Documentation for the Benchmark Driver 34
   B. Documentation for the Workload Analyzer 73
   C. Documentation for the Workload Extractor 93
   D. Documentation for the Subworkload Analyzer 112

Acknowledgements

I am deeply indebted to many of my professors and colleagues who have helped me with this project in their various ways. In particular, I wish to extend my sincere appreciation to Domenico Ferrari, whose gentle guidance has provided the ground upon which preliminary efforts could grow into a Research Project in this area; to Al Despain, whose willingness to review this project on short notice was much appreciated; to Bob Holt, who participated in the original design of the Benchmark Driver; to Bob Schulman, my UNIX * guru, who always seemed ready and willing to answer my questions about UNIX and the C Programming Language; and to the Progres Group, whose feedback at various points in the project was quite supportive.

In addition, this research was supported in part through the generosity of Bell Telephone Laboratories, and by the National Science Foundation under grants MCS78-24618 and MCS78-07291.

* UNIX is a trademark of Bell Laboratories.

Tools for Research in Computer Workload Characterization and Modeling

1. INTRODUCTION

The digital computer has received increasing acceptance as a tool with which to handle many of the tedious tasks required of human beings. This increased acceptance of computers has brought about a proliferation of computers of various size and capacities. With the increased number of computers on the market, and with the corresponding increase in the amount of computer software, the need to effectively compare the performance of various hardware/software configurations becomes more acute. This need becomes more acute both in computer procurement and system improvement studies. It becomes more and more important that we be able to easily answer questions such as: "will a given change in this piece of hardware, or this computer software, be beneficial?"

Unfortunately, the current state of the art of Computer Systems Performance Evaluation is such that there are no methodological approaches to answer questions such as these. Each new question in Computer Systems Performance Evaluation demands a unique combination of techniques on the part of the investigator. In attempting to answer a question such as: "will the computer to its job faster if this piece of hardware is changed," the first question to be asked is: "what is the computer's *job*?"

Making an adequate evaluation of a system requires examining its performance as it processes its workload. It is often difficult to evaluate a system as it performs its day-to-day tasks, or it's "real" workload. It is usually desirable to monitor the computer's activity as it processes a more compact "model" workload. Problems arise, however, because there is currently no agreement on how to construct a workload model. There are several approaches to the workload modeling problem, and each approach has its advantages and disadvantages. No approach is viewed as superior to the others.

One approach, however, shows the potential to solve many of the problems with workload modeling. The performance-oriented approach uses performance criteria as a guide to constructing a workload model. This approach shows promise, but it also shows a need for further refinement. The purpose of a research project at the University of California, Berkeley, was to examine these needs and to plot a course for developing the performance-oriented approach further. Three questions were asked in the course of this investigation:

1. What are the problems in workload characterization and modeling, and what solutions does the performance-oriented approach have to offer?
2. What are the needs for further research in the performance-oriented approach, and what software tools would be needed to support this research?
3. To what specific uses could these tools be put?

This project succeeded in defining the needs for further research, constructing tools to support this research, and plotting future courses of action.

The purpose of this paper is to present the results of this investigation. The workload modeling problem is examined, and the needs for further development of the performance-oriented approach are defined. Tools are needed to support future research in this area, and the requirements of four such tools are outlined. These tools were constructed as a part of the project, and their function and operation are described. The set of tools described in this paper do more than support general efforts in workload characterization and modeling; they have direct applicability to a specific experiment that would serve to further develop the performance-oriented approach. Before turning to this experiment, however, let us take a more general and thorough look at the workload modeling problem.

2. THE WORKLOAD MODELING PROBLEM

When making a comparison of two computer systems, the most frequently asked question is: "which system performs better?" After scratching the surface of such a question, it is found that the underlying question is: "which system performs its tasks better?" Whether the systems being compared differ in hardware, software, or both, the performance of two systems must be compared with respect to the tasks they must perform, or the workload that they must execute. The question of computer systems performance cannot be answered independently of the system's workload.

When two computer systems are being compared, they must be compared while executing the same workload. If the workload on the systems being compared is not held constant then the performance indices, being dependent on the workload, are meaningless. Ideally, the workload executed in a performance study should be the same workload as the system executes in its daily use. The workload processed in an industrial environment will vary over periods of a day, week, month, quarter, and fiscal year. Similarly, a workload processed in an academic environment will vary over time periods imposed by the academic calendar. It would be desirable then, to base a performance comparison on a workload having a duration of a fiscal or academic year. Such a comparison would examine system performance in response to all possible variations in the workload, and hence the performance indices under examination would be valid. Conducting a performance evaluation that requires a year to collect the data, however, is obviously infeasible. What is needed is a smaller, more compact workload that *models* the system's real workload.

The basic requirement of a workload model is that it accurately *represent* the real workload being modeled. That is, the workload model must be equivalent to the real workload for (at least) the parameters of interest in the study. The difficulty in workload modeling is that at present, no tried and true methodology exists for making a representative model of a real workload. This problem stems from the fact that there is no agreement on what the elusive term "representative" means. Different definitions of "representativeness" determine different approaches to the workload modeling problem. There are advantages and disadvantages to each of the approaches to workload modeling. One of the approaches, however, may have the potential to overcome its drawbacks.

2.1. The Resource Consumption Approach

The most often selected approach to the workload characterization problem is the resource consumption-oriented approach. This approach is based on the premise that a model workload is representative of a real workload *if they consume the same resources at the same rate.* A workload model constructed following a resource consumption approach might be characterized in terms of CPU time used, memory space required, and frequency and duration of I/O operations. Although the model workload does not consume the same quantity of resources as the real workload, the model workload (hopefully) exhibits the same consumption patterns as the real workload. For instance, it would be desirable for a model workload to maintain the same ratios between resources consumed as the real workload. As well as providing a criterion for constructing a model consisting of terminal scripts, the resource-consumption approach provides the rationale for constructing synthetic workload models. There are two major drawbacks with the resource consumption approach, however, which leave the workload model with much to be desired.

First, the model is very sensitive to the choice of resources to be consumed and the patterns of resource consumption to be followed. The choice of resources to be consumed is critical. The choice must span the set of computer resources, and the choice must not contain redundancies. For instance, specifying CPU time for a model is incomplete where different instruction mixes cause different performance characteristics to be exhibited. As well as specifying the amount of memory consumed, for instance, the pattern of memory references must also be specified. The pattern of memory references in a virtual memory system can often be more important than the gross amount of virtual memory used. After examining any resource consumption specification, it becomes clear that specifying the resources and their consumption patterns is not as easy as it may appear on the surface.

Second, the resource consumption approach suffers in its dependency on a particular machine and configuration. Resource consumption specifications may be interpreted only in the context of the machine on which they are defined. Consider, for instance, a specification that depends on disk access time. If the physical characteristics of the disk are changed, or even if the files are re-arranged on the same disk, the observed performance of the model workload is likely to change. Consider a workload model that specifies CPU time down to the instruction mix and sequences. These specifications have little meaning when transported to a machine having a different instruction set. Both of the drawbacks of the resource consumption approach focus on problems defining the workload with enough detail, yet with enough abstraction to maintain machine independence. This points to a need for a higher-level definition of representativeness with which to drive a workload modeling methodology.

2.2. The Functional Approach

The functional approach to workload characterization employs a definition of representativeness higher than the resource consumption approach, thereby solving some of the problems inherent in the latter. This approach is based on the premise that a model workload is representative of a real workload if the model performs the same *functions* as the real workload. If the real use of the computer is to do payroll processing, then the model workload should do payroll processing. If the real workload includes program compilations, then the model should perform program compilations. The process of constructing a workload model becomes simply reducing the number and size of the tasks executing any of the "representative" functions. The number of tasks performing payroll processing or program compilation (for example) are reduced, and the size of the payroll and the length of the programs to be compiled are reduced. This higher-level definition of representativeness appears sound, but it too has its difficulties.

In attempting to raise the level of detail from that of the resource consumption approach, the functional approach merely transforms the question of representativeness into another domain. Rather than specifying the actual programs and commands to be executed in the workload, *all* are used, except with a reduction in the size of the input to the programs. What remains is to reduce the size of the inputs used in the model workload. The inputs must be reduced in such a way that they are representative of the inputs encountered in the real workload. But this is no less complicated than the original problem of selecting a representative *workload*. How, for instance, is a "representative" subset of payroll data, or programs to be compiled, to be selected? Should a representative subset cause the same code paths to be executed? Should the subset cause the same resources to be consumed at the same rate? It is clear that the functional approach merely shifts the question of representativeness from the actual processing done to the data used by the processing, leaving few problems solved along the way.

2.3. The Performance Oriented Approach

A third approach to the workload modeling problem again transforms the problem of representativeness, but in a more constructive manner than the functional approach. The performance-oriented approach to workload characterization is based on the premise that a model is representative of a real workload if the executing model causes the system to exhibit the same performance characteristics as the real workload. A workload model constructed with the guidance of a performance-oriented approach would be calibrated in such a way that it puts the subject system through the same "paces" as the real workload. The goal of a performance comparison is presumably to examine different systems under all of the load levels that they are expected to encounter. A performance-oriented approach to workload design would thus well serve the purposes of such a performance comparison. Although this approach, like the others, has its own drawbacks, the performance-oriented approach has the inherent quality that it guides the construction of the workload model.

The performance-oriented approach is essentially a black-box approach to workload modeling. Rather than being concerned with the fine details of what goes on *inside* the workload model, the performance-oriented approach emphasizes what goes on *outside* the model. What is really of importance is the performance that the workload induces on a system, which is what the outside world sees. A black-box approach to computer systems performance evaluation is an approach quite consistent with the rest of the engineering world. The digital computer system, being a man-made machine, is a part of this engineering world, and perhaps we would benefit by approaching it from this perspective.

One possible drawback to the performance-oriented approach might be that, like the functional approach, it merely transforms the question of representativeness into a different domain. The functional approach was criticized for transforming the question of representativeness from "what programs are representative?" to "what input data are representative?" Similarly, one might criticize the performance-oriented approach for transforming the question of representativeness from "what programs and data are representative?" to "what performance indices are of importance?" Since a workload is considered "representative" if its performance indices match those of the original workload, it might appear that emphasis is shifted away from the construction of a workload towards choosing appropriate performance indices. But this change in emphasis is precisely the change in emphasis needed to define a workload modeling effort!

Rather than allowing the workload characterization process to become bogged down in the details of what resource consumption rates are representative, or what program inputs are representative, it forces the characterization process to focus on the variables *of interest in the performance study*. For instance, if interactive services are to be compared, the performance indices of interest would necessarily relate to the responsiveness of the system (Ferrari, 1979a). If, for instance, two disk scheduling algorithms are to be compared, performance indices of interest would relate to disk access time and disk queue lengths. Once the needs of a performance evaluation are defined in terms of the performance indices of interest, the criterion for determining the representativeness of a workload model are automatically established. In the context of a given study, the model workload is representative if the performance indices of interest match those of the real workload. Thus the needs of the performance evaluation efforts guide the construction of a model workload which, in turn, aids in the collection of the data of interest in the performance evaluation study. If the performance indices used to calibrate the workload model are more comprehensive than the indices required by a given study, the validity of the model is likely to be enhanced and made applicable to other evaluation efforts as well (Ferrari, 1978).

A second potential criticism of the performance-oriented approach is that, like the resource consumption approach, it lacks in machine independence. The performance-oriented approach provides guidance in constructing and calibrating a workload model from a real workload, but what if the workload model is to be transported to a different machine? If the second machine hosts the same programming languages and system commands, the workload may be transported quite simply. If the second machine is only similar to the original, a functional approach may be taken, and the system commands could be translated to perform the same functions on the second machine. If the difference between the two machines is fairly drastic, a combination of functional and resource consumption approaches might be taken in transporting the workload model. Little real application has been made of the performance-oriented approach, so little has been done towards solving problems such as this. It is probable that further insights will be gained as the performance-oriented approach receives more attention in the literature.

The performance-oriented approach to workload modeling has received some attention, but studies which adequately examine its validity and practicality are non-existent. Nolan and Strauss (1974) make an early attempt at using performance criteria for building a workload model on the Xerox Sigma-7/UTS timesharing system. Their efforts, however, are more directed towards the *construction* of the workload model than towards *evaluating* its representativeness.

Ferrari (1979a) presents a precise methodology for constructing a workload model from a real workload. This methodology outlines the modeling process from the characterization of the workload to the verification of the model. The emphasis of this effort, however, is placed in the development of the methodology, and does not put it to a practical test of validity. Both of these papers provide direction for a performance-oriented workload modeling effort, but neither approach is proved to be valid nor shown to be practical.

It is clear that the need at present is for investigation into the validity of the performance-oriented approach. The reason that Nolan and Strauss (1974) and Ferrari (1979a) fall short of this need is that they both lacked the *tools* with which to test their methodologies, and to make quantitative assessments of their validity. What is needed in each case is a set of tools with which to construct a model workload and assess its validity. The performance-oriented approach to workload modeling shows more promise than the resource consumption or the functional approaches. Thus efforts toward developing tools for further research in the performance-oriented approach would be well placed.

3. TOOLS FOR WORKLOAD CHARACTERIZATION AND MODELING

An examination of the problems in workload characterization and modeling has revealed that the performance-oriented approach shows promise. The performance-oriented approach guides the modeling effort through its definition of representativeness. A workload modeling methodology that provides this guidance may prove to be quite practical. Questions regarding the validity of the approach have been raised, and preliminary answers have been given. The answers to the questions surrounding the performance-oriented approach cannot be answered without having the experience of applying it. It would be desirable at this point to apply a performance-oriented workload modeling methodology to construct a workload model, and to follow through on the analysis required to answer the questions raised about it. More precisely, the analysis which is needed at this point is the comparison of a workload to its model from the performance point of view.

Unfortunately, the reason that this analysis has not yet been done is that there are no tools for research in this area. Many systems provide a means by which a workload may be artificially executed, but on no system does there exist the comprehensive set of tools that such an investigation requires. A first step necessary to allow such an investigation to take place, then, is the construction of tools for research in workload characterization and modeling.

What are the requirements for tools to support experimentation with the performance-oriented approach to workload modeling? The most important tool would be one that allows a workload to be placed on a system. As well as providing the ability to apply a workload to the system, this tool would also be required to collect data on the performance of the system as the workload executes. A second requirement is for a tool that will, once the performance data is collected, analyze the data in such a way that the behavior of the workload may then be characterized. Once the activity of the workload is characterized, it would be desirable to extract representative subsets of the real workload from which to construct the model workload. A third requirement is for a tool that would extract portions of a real workload for inclusion in a model. The fourth and final tool that would be required is one that, given performance data from the execution of the model workload, analyzes the data. The goal of this tool would be to answer the question: "is the model workload representative of the real workload?" Clearly, answering this question would be the goal of research in workload modeling.

The goal of a research project at the University of California, Berkeley was to develop the specifications for tools to satisfy the above requirements, and to construct those tools. In this way, the preliminary needs of research in workload modeling could be met, and the door would be opened for further investigation into the performance-oriented approach. The four tools whose function is briefly described above were deemed necessary to support the needed research. The Benchmark Driver provides the ability to apply a workload to a UNIX * timesharing system. The Workload Analyzer provides statistical summaries of performance data for use in the characterization process. The Workload Extractor provides the means by which representative portions of the real workload may be used in the construction of a model workload. Finally, the Subworkload Analyzer analyzes the performance data from execution of the model workload for use in answering the question "is the model representative of the real workload?" The function and features of each of the tools is described in the following pages, and more complete details of the use, structure, and the program listings of the tools may be found in the appendices.

* UNIX is a trademark of Bell Laboratories.

3.1. The Benchmark Driver

The UNIX Benchmark Driver is designed with two basic needs in mind: the need to be able to place a *repeatable* workload on a machine, and the need to easily *modify* the workload. A *repeatable* workload is needed in order to perform experiments with reproducible results. An easily *modifiable* workload is easy to tune in order to meet some modeling criterion. When used to develop a workload model under a performance-oriented approach, the Driver executes interactive terminal scripts and produces performance statistics as output (see Figure 1).

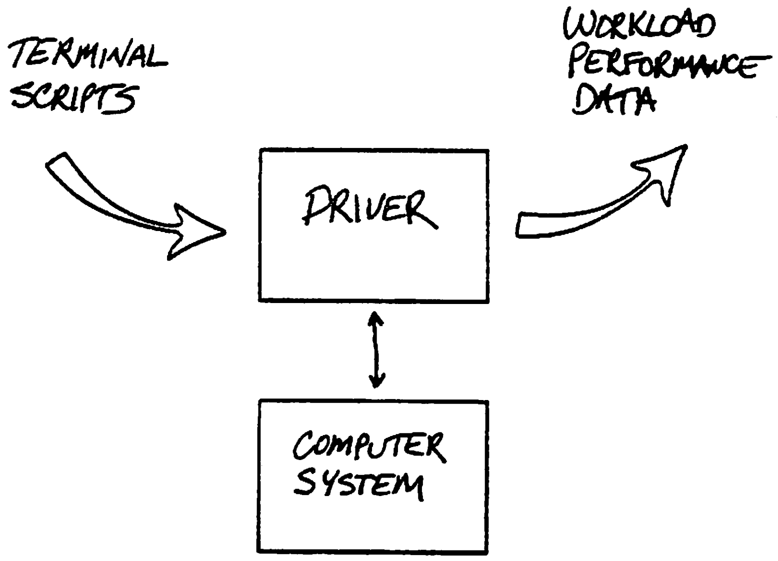

Figure 1: Function of the Benchmark Driver

The Driver places a repeatable workload on the system by preparing a number of driver command scripts, and starting a number of concurrently executing processes that read commands from the script files. The workload imposed on the system is made easily modifiable by allowing an analyst to interactively specify the number of concurrent processes (or simulated "terminals") to execute, and which script files each process is to execute.

The Benchmark Driver assembles a driver script file for each simulated "terminal" in response to commands typed by the system analyst (see Figure 2).

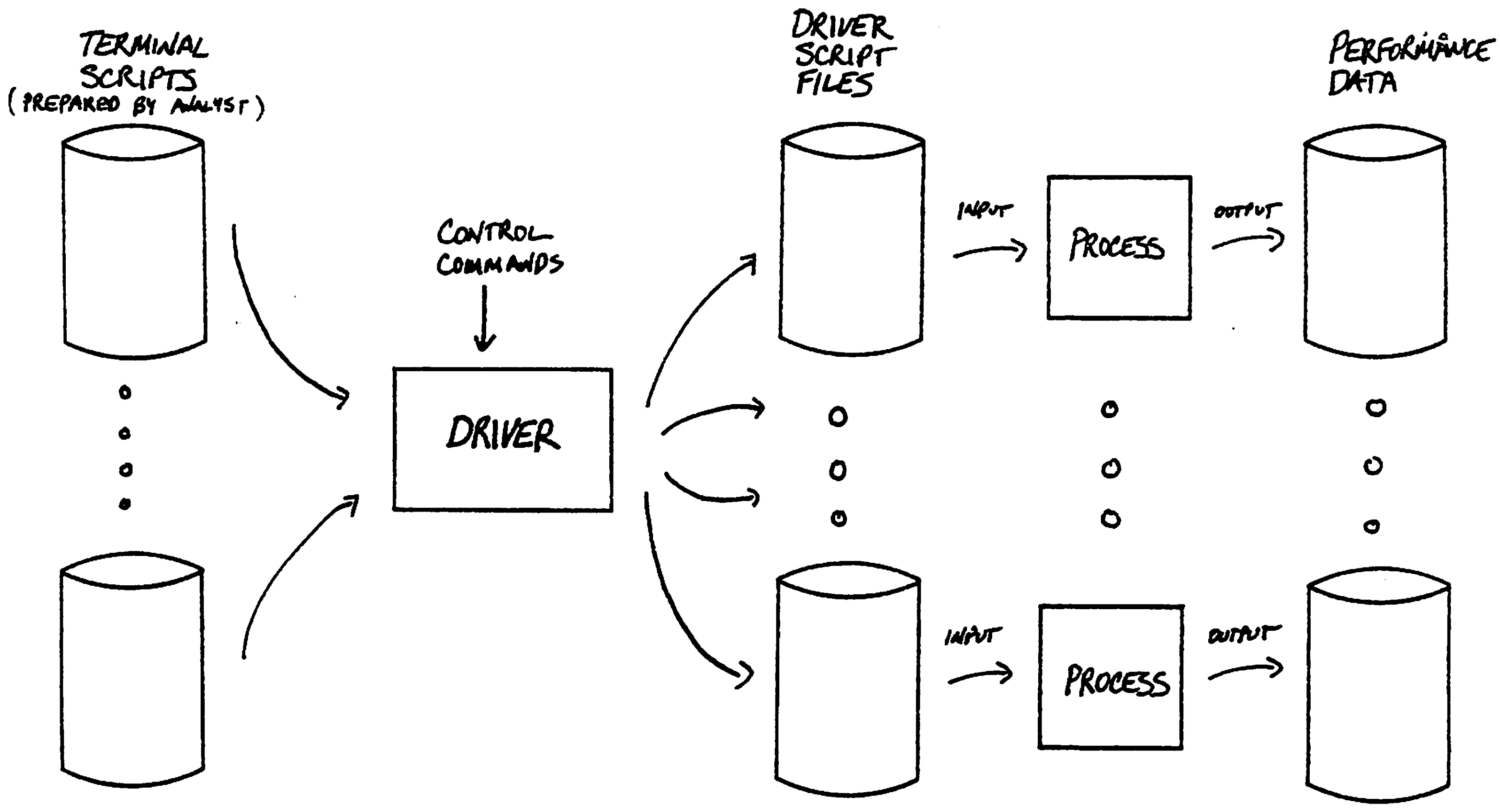

Figure 2: Operation of the Benchmark Driver

It basically takes a number of shorter script files prepared by the analyst and repeats them in a file to be used as input to a process. One file is prepared for each terminal to be simulated. If more than one simulated terminal is to execute the same set of script files, the Driver will optionally permute the order of files executed by each simulated terminal, thereby avoiding placing a completely homogeneous workload on the system. An analyst might, for instance, prepare scripts containing commands for an editor session, a script for program compilation, and a script which executes several user programs. The analyst might indicate to the Driver that 20 terminals are to execute the edit script 10 times, the compilation script 5 times, and the program execution script 15 times. Twenty driver script files, one for each terminal, would be created in such a way that the ordering of the 30 files specified by the analyst would be different in each file. The Benchmark Driver would then proceed to start 20 processes, each process reading commands from one of the script files created by the driver. Although the construction of driver script files sounds simple, the files themselves are actually a bit more sophisticated.

The files created by the Benchmark Driver contain extra information to gather performance data for the benchmark run, and to increase the "realism" of the simulated terminals. The file (optionally) contain commands at the beginning and end of the driver script which enable the statistics-gathering capabilities of the system. In order to simulate the time that a terminal will sit idle as a user "thinks" between commands, random waits are inserted between each command in the files created by the Driver. The wait times are simulated by issuing a UNIX.I sleep command. This function of the Driver may be activated or deactivated by commands placed within the script files prepared by the system analyst. The distribution of the random "think" times is under control of the analyst. The distribution may be either uniform or exponential, the parameters of which may be set by the analyst.

One of the most important requirements of the Benchmark Driver is that it collect performance data from each simulated terminal, and this requirement was not neglected in its design (see Figure 2). System performance data is collected for each command executed by the Driver. A unique time stamp indicates the completion time of the command. CPU time statistics indicate the user and system time for the command. The response time for the command is given along with the time expansion factor, indicating the percentage of elapsed time during which the CPU was also processing the command. In order to collect this data, the Benchmark Driver executes a UNIX *shell** and editor that have been modified to produce the data for every command processed. In addition, the modified *shell* and editor also execute the *sleep* command (used to simulate "think" times) in such a way that no performance data is collected for this "command." Through mutual cooperation between the Driver, the *shell*, and the editor, the Benchmark Driver is able to collect the performance data deemed necessary to characterize the workload.

Compromises between realism and practicality were (of course) necessary in the design of the Benchmark Driver. A driver that would be almost ideal would be an external computer that transfers commands to the subject computer via the latter's own I/O lines. Driving a system in this manner would be ideal because there would be little (if any) difference in the system's response to commands "typed" by the driver computer and commands typed by the real users. Even the code paths executed within the operating system would be the same. Such an approach, however, is not always practical, and the simpler internal driver was chosen for this collection of tools. Compromises such as these are necessary in any effort such as this. The design decisions followed by others, along with the compromises made are often enlightening. The interested reader is referred to descriptions of other benchmark drivers, particularly those described by Abrams and Treu (1977), Fogel and Winograd (1974), Turner (1976), and the University of Michigan Computing Center (1978).

With the ability to drive a UNIX system with a prepared workload established, other tools are needed to support the remainder of the workload characterization and modeling tasks.

* The *shell* is the command interpreter used in the UNIX time sharing system.

3.2. The Workload Analyzer

Once a workload is applied to a system and performance data from its execution is gathered, it is desirable to summarize the data. Many lines of performance data are useless without the capability to reduce the data into a manageable summary. Once the performance data is summarized statistically, the workload may be characterized in terms of the performance that it induces on the system. The Workload Analyzer provides the facility to examine the performance data produced by the Benchmark Driver, and produces the statistical summary needed to characterize the workload (see Figure 3).

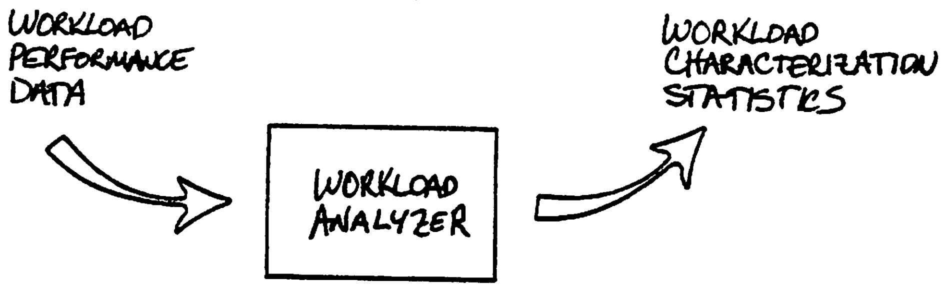
WORKLOAD PERFORMANCE DATA
WORKLOAD ANALYZER
WORKLOAD CHARACTERIZATION STATISTICS

Figure 3: Function of the Workload Analyzer

The Workload Analyzer partitions the workload into chronologically sequential "subworkloads," and provides a statistical summary of each subworkload. A subworkload is simply defined as the "next" n interactions processed by the system, where "n" is a parameter of the Analyzer. Thus a subworkload is a quantum of time in the execution of the workload, where "time" is measured in commands executed (in contrast to minutes and seconds). The Workload Analyzer partitions the workload into subworkloads by examining the output files from each of the simulated terminals, and selecting the statistics from each in a strict chronological order. A separation is made between subworkloads every time "n" statistics lines have been examined. The Workload Analyzer also indicates which statistics lines were used from each simulated terminal. A workload viewed in terms of subworkloads allows it to be viewed in small segments, the performance indices of which each have the same statistical significance. A workload examined in this manner readily displays the different "paces" through which the workload puts a system. Once the various subworkloads are examined in terms of the performance that they induce on the system, the more representative ones may be selected for inclusion in a workload model. Once the representative subworkloads are selected for inclusion in a workload model, a tool is needed that will extract the subworkloads from the real workload.

3.3. The Workload Extractor

The Workload Extractor is a tool that will extract portions of a real workload for inclusion in a model workload. The Workload Extractor assumes that the "real" workload was applied to the system with the Benchmark Driver, and that the driver script files still exist. The Workload Extractor acts as an editor that, under the user's direction, selects lines from a driver script file and places the extracted lines in a file to be used late as a driver script file (see Figure 4). Given a set of subworkloads that are chosen as "representative," a model workload is constructed by extracting the commands comprising each subworkload from each driver script file, one file at a time.

Although it may appear on the surface that a standard line editor may be used to perform this extraction, the functions of the Extractor are actually quite a bit more complex. The Analyzer, since it examines only the statistics lines

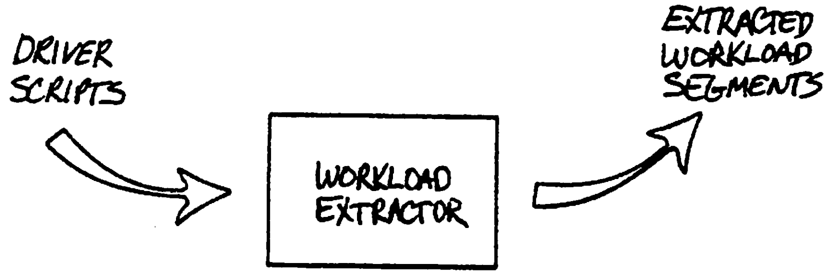

Figure 4: Function of the Workload Extractor

provided by a simulated terminal, counts as commands only those commands which produce a performance statistic line. There are many commands which do not produce statistics lines, most obviously the *sleep* command used to simulate user "think" times. Therefore, when the Analyzer indicates which lines from which terminals are included in a subworkload, the line number ranges given are not lines in the driver script file, but ranges of command lines *for which statistics are produced.* The Extractor, given these command number ranges, must know which commands cause statistics lines to be generated, and handle the extraction process accordingly. Additionally, the Extractor must know about special commands, such as the edit command. Suppose, for instance, that one terminal's contribution to a subworkload is five lines from the middle of an editor session. Blindly extracting these five command lines from the driver script would place editor commands amidst shell commands. The Extractor handles problems such as this by ensuring that the editor is properly entered and/or exited in these cases.

The Benchmark Driver allows a workload to be placed on a UNIX system, the Workload Analyzer allows the workload to be characterized by representative subworkloads, and the Extractor allows the representative subworkloads to be assembled to form a model workload. So far, this collection of tools makes it easy to construct and execute workload models, but it does not provide a means to analyze the performance of the workload model as a modeling methodology may require.

3.4. The Subworkload Analyzer

The Subworkload Analyzer provides an alternative means to analyze the performance of the model workload other than with the Workload Analyzer. It may be recalled that the Workload Analyzer classifies subworkloads as sequences of commands as they are executed in chronological order. Once a workload is modeled with a set of subworkloads, and the workload model is executed, it may happen that the commands are not executed in the model in the same order in which they were executed in the real workload. Hence the lines demarking each subworkload become fuzzy, as does the definition of a subworkload.

A short example may help to clarify this point. Suppose, for instance, that a real workload on a system involves two terminals processing interactive commands. A model workload was constructed by extracting two subworkloads of 100 commands each from the real workload. The first subworkload consists of commands 1 through 75 in terminal 1, and 1 through 25 in terminal 2 (see Figure 5). The second subworkload consists of commands 76 through 125 in terminal 1, and 26 through 75 in terminal 2. The subworkloads as included in the model sare separated by the double line. Suppose that, after executing the model workload, the first 100 commands processed are commands 1 through 71 from terminal 1 and 1 through 29 from terminal 2. This is indicated by the dashed line. There are still 100 commands in the first subworkload, but not the 100 commands that were extracted to create the first subworkload. The first subworkload in the model no longer matches the corresponding subworkload in the real workload. Because the commands executed in the subworkloads are not the same, the performance of the model cannot be compared to the real workload. It might be desirable, then, to artificially maintain the original "boundaries" between subworkloads when creating the model.

The Subworkload Analyzer provides the capability to maintain a one-to-one correspondence between the commands comprising a subworkload in the real workload and the commands executed in the model workload. Thus the statistics generated by a subworkload in the real workload (as analyzed by the Workload Analyzer) will be generated by the same commands in the model (as analyzed by the Subworkload Analyzer). The Subworkload Analyzer (see Figure 6), rather than accumulating statistics in chronological order, extracts statistics lines from each driver output file until a flag indicating the end of a subworkload is reached. Operation of the Subworkload Analyzer is supported by the Workload Extractor, which places a flag between each set of lines extracted at the direction of the user. With the aid of the Subworkload Analyzer, the performance of the model workload may be compared to that of the real workload, and the question: "is the model representative of the real workload?" may be answered.

This box of tools for workload characterization and modeling is thus filled with four tools. The Benchmark Driver allows a workload to be imposed on a system. The Workload Analyzer aids in characterizing the workload. Once representative subworkloads are selected, they may be removed from the real workload with the help of the Workload Extractor. A model workload may be

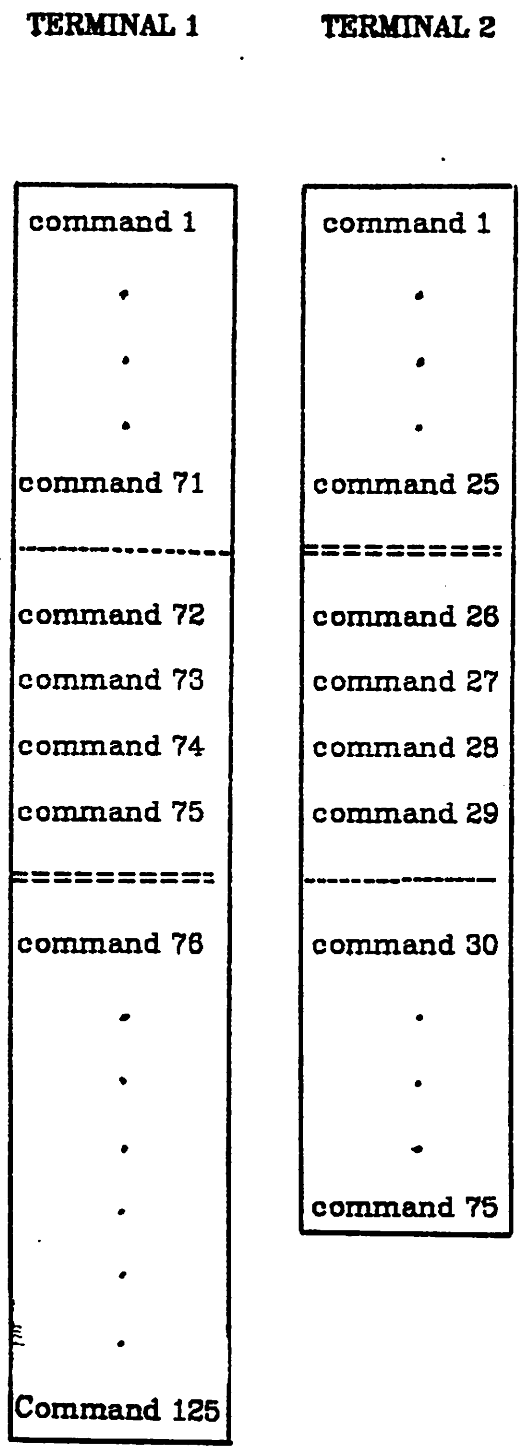

Figure 5: Maintaining the Original Boundaries Between Subworkloads

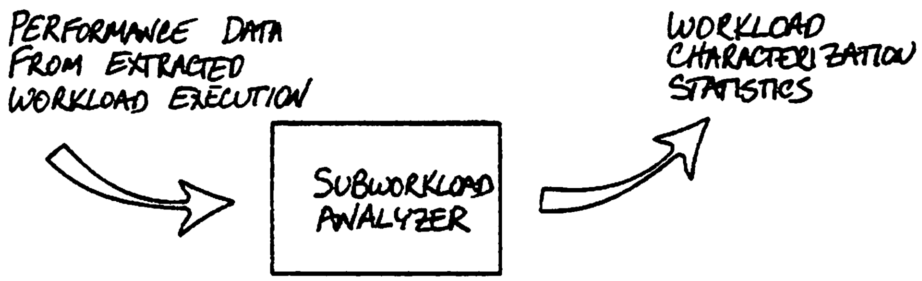
PERFORMANCE DATA FROM EXTRACTED WORKLOAD EXECUTION
SUBWORKLOAD ANALYZER
WORKLOAD CHARACTERIZATION STATISTICS

Figure 6: Function of the Subworkload Analyzer

constructed from extracted subworkloads, the performance of which may be examined by the Subworkload Analyzer. Clearly this set of tools is the comprehensive facility needed to assist in research in the practicality and the validity of the performance-oriented approach to workload modeling.

4. USE OF THE CHARACTERIZATION AND MODELING TOOLS

No collection of tools would be complete without an owner's manual, and it would be unwise to leave the reader without suggestions for using the tools described in this paper. Four tools useful in workload characterization and modeling efforts have been described, and it is timely to propose some uses to which these tools might be put. The four tools presented in this paper support five phases of a workload modeling effort. The tools facilitate placing a "real" workload on a UNIX system, characterizing that workload, extracting representative portions of the real workload, placing the model workload on the system, and evaluating the model's performance. The approach to workload characterization and modeling supported by these tools is the performance-oriented approach. The tools give feedback on the workload's performance, rather than its function or its patterns of resource consumption. The ideal use for this set of tools, thus, is for further research in performance-oriented workload modeling methodologies. As was the earlier conclusion, this is an area in the field showing promise, and it is deserving of further exploration.

A particularly useful application of the set of tools described in this paper is to study and attempt to verify a performance-oriented workload modeling methodology. One such methodology was recently proposed in a paper by Ferrari (1979a). Ferrari proposes a methodology based on the performance-oriented definition of representativeness. A model is representative of a real workload if the model *induces the same performance on the system as the real workload.* Ferrari's methodology begins by characterizing the real workload in terms of the performance that it induces on the system. For the comparison of interactive systems, performance indices of interest might be command response times and the variance from the mean of response time. Whatever the choice of indices, the performance of the real workload is examined by subworkloads, each subworkload having a length on the order of 100 interactions. Each subworkload has associated with it a value for each of the performance indices of interest. The set of performance values for each subworkload may be considered as coordinates in a performance "space." One point in the performance space represents a single subworkload. With all of the subworkloads represented in this space, a clustering algorithm is used to define sets of subworkloads whose performance characteristics are similar. One subworkload is then chosen as a representative from each cluster. The point closest to the center of mass of each cluster is selected. A model workload is constructed using the interactions from each subworkload chosen as a representative of a cluster. The model is executed and performance data from each subworkload is collected. The values of the performance indices for each subworkload are weighted to account for the size of the clusters from which they come. The answer to the question: "is the model representative of the real workload?" is answered by comparing the weighted performance indices from of the model with the aggregated performance indices from each cluster in the real workload. The model is considered representative of the real workload if the performance induced by the model is, within some error, equal to that of the real workload.

The workload characterization and modeling technique proposed by Ferrari is consistent with the performance-oriented definition of representativeness. Because of its novelty, however, many questions concerning its use and validity remain. First and foremost, will the methodology produce a representative model workload? If not, might the methodology be modified so that it will? What clustering technique is best for choosing representative subworkloads? What is a good scale for the model, *i.e.*, how many subworkloads in the real workload should be represented by one subworkload in the model? Preliminary answers to these questions may be obtained by using simulation models, but the real test of practicality and validity of the technique is to apply it on a real system.

4.1. A Workload Modeling Experiment

Clearly, the tools presented in this paper provide the means to put this workload modeling methodology to a live test. In the process of actually using and experimenting with the proposed methodology, the answers to the above questions may be found. Therefore, we wish to propose an experiment with which the technique proposed by Ferrari (1979a) may be validated and possibly improved. There is one underlying assumption in the experiment which makes it practical to execute in a controlled environment. It is assumed that, if the workload modeling technique works to make a model from a real workload, then it will also work to make a model of a simulated workload. With this assumption it is possible to construct from scratch a workload to be considered "real" and then proceed to make a model of it. This will allow attention to be focused on the modeling process rather than monitoring a real system in execution. The procedure for this experiment, then, involves following the six steps outlined below.

1. Use the Benchmark Driver to prepare and execute a workload that is to be considered the "real" workload. In order to produce approximately 250 points in the performance "space," with 100 interactions per subworkload, a total of 25,000 commands would be required. Variation should be emphasized in developing the "real" workload so that both "light" and "heavy" use of the machine is simulated. The 25,000 commands could be executed by anywhere from 10 to 50 terminals on the VAX 11/780 computer*. Experimenting with the number of simulated

* VAX is a registered trademark of Digital Equipment Corporation. The VAX 11/780 is the computer used for research computing in the Computer Science Division, EECS Department at the University of California, Berkeley.

terminals may yield some interesting results.

2. Use the Workload Analyzer to obtain performance data on the workload. If 100 interactions per subworkload are used, 250 subworkloads will be obtained. 250 points in the performance space should provide enough points to produce a significant number of clusters.

3. Locate approximately 20 to 25 clusters in the performance space. This would cause 20 to 25 subworkloads to be included in the model, which would be a reasonable ten-to-one reduction in workload size. The point (or subworkload) closest to the center of mass of each cluster should be chosen as "representative." A simple euclidian clustering algorithm may be used, or more sophisticated approaches might be taken. The interested reader is referred to Diday (1979).

4. Use the Extractor to construct a workload model consisting of the 20 to 25 subworkloads that were representative of the "real" workload. The driver script files are used as input to the extractor, and ready-to-execute scripts are the output.

5. Execute the Model Workload either by using the Benchmark Driver, or by manually starting up the appropriate number of *shells*.

6. Analyze the execution of the model workload with the Subworkload Analyzer. The weighting procedure described by Ferrari (1979a) is used to arrive at performance values for the "real" and the "model" workload. Once these values are obtained, the performance induced on the system by the model workload may be compared to that of the "real" workload. The closeness to which the two sets of performance indices match indicates the degree to which the model is representative of the real workload.

One of the major benefits of such an experiment is expected to be the opportunity to tinker with the methodology and to tune some of its parameters. The fine details of the experiment are left vague because they are simply not known at this time. The details not explicitly specified in the experiment are left open so that the experiment may focus on finding out "what works best." For instance, the numerical values specified above were estimated by Ferrari (1979b), and they aim for a 10:1 reduction in size from the real to the model workload. It may turn out that a larger or smaller modeling scale works better, so changes in the parameter values should be tried. Step 5 in the experiment, for instance, does not specify whether the subworkloads making up the workload model should be executed independently, or one after another in one combined run. If the subworkloads are executed independently, there will be no transition problems between subworkloads. It takes processing several commands before a computer system reaches a stable level of performance. If this startup transient is significant, it will affect the performance of each subworkload, if executed one at a time. Similarly, if the subworkloads are executed one after another, a performance transient will be encountered as the system finishes executing one subworkload and begins processing the next. It may be discovered that the startup transient in performance is greater than the transient encountered when passing from one subworkload to the next, and it is better to execute the subworkloads in the model one after the next. Thus changes in the experimental procedure should also be tried in order to find out which techniques work best. The major thrust of this experiment should therefore be to improve and tinder with the workload modeling methodology, rather than simply to attempt to prove or disprove its validity.

An experiment such as this is essential to further research in performance-oriented approaches to workload modeling. A methodology cannot be of general use until some experience with its use gained and its details are determined. Performing the experiment proposed above would increase our body of knowledge in this area, and it would provide a means with which to define more precisely the procedures in the modeling methodology.

5. CONCLUSION

The increasing popularity of digital computers has made more urgent the need to evaluate their performance. The first step in any performance evaluation study is to determine what tasks the computer is to execute as its performance is being evaluated. This step usually involves making a model of the real day-to-day workload placed on the computer system. Making a model workload involves characterizing the activity on the system and constructing a model which is *representative* of the real workload. This is where the problems in workload modeling begin: different definitions of what is "representative" bring about different approaches to workload modeling. The resource-consumption approach to workload modeling suffers in its dependence on exact specifications of the resources to be consumed and their rates of consumption. The functional approach suffers in its dependence on a representative model of the computer system's inputs. The performance-oriented approach might be said to suffer from its dependence on a correct selection of performance indices, but it was shown that this actually guides, rather than hinders, the workload modeling effort.

What the performance-oriented approach suffers from the most is a lack of attention. The papers on the subject propose methodologies based on the performance-oriented approach, but they do not actually evaluate the modeling techniques based on actual experience. Tools are needed to support further investigation in the performance-oriented approach. The purpose of this paper is to explore the requirements for tools to satisfy these needs, and to describe the tools developed to meet the requirements. The Benchmark Driver provides the capability to load a UNIX system with a repeatable, easily modifiable workload. The Workload Analyzer analyzes the system's performance induced by the workload, and the Workload Extractor provides the capability to extract portions of the workload for stand-alone execution. Finally, the Subworkload Analyzer provides statistics on the execution of extracted portions of a workload.

These tools do more than support a general need for further investigation; they support specific applications in a workload modeling experiment. The four tools described in this paper find direct application in an experiment designed to further develop a recently-proposed workload modeling methodology. These tools do well in supporting current efforts in this field, but to what use might they be put in the future? What possible developments may be seen in the future of workload characterization and modeling?

The digital computer, being a man-made machine, is a part of a more general field of engineering, and perhaps we would benefit from viewing it in this context. At one point in this presentation we alluded to performance *transients*, as if the computer were just another one of the many dynamic systems found in engineering. Just as an electronic circuit requires time to reach a steady-state or a chemical reaction requires time to reach an equilibrium, a computer system requires time to respond to a sudden change in its workload. When a computer system is subjected to a sudden change in tasks to perform, its performance does not instantaneously jump to some constant value. Just like any other dynamic system, a computer undergoes a transition from one performance level to another. If a change in its workload *(input)* causes a change in its performance *(output)*, perhaps a computer system also has a transfer function that determines its response to a given workload (Ferrari, 1979b).

Once again, properties such as this might be explored using the tools for workload characterization and modeling presented in this paper. The tools provide the capability to place a workload on a machine, the capability to place a portion of the workload on the machine by itself, and the ability to analyze the response of the machine to the changes in workload. In the abstract, the tools provide the ability to apply different inputs to a system and to measure the system's output in response to the changing inputs. This would be a first step in searching for a computer system transfer function. Being able to describe the performance of a system in such a straightforward and simple way would open up many new perspectives on the computer system. If a computer system could be viewed as any other system in the engineering world, the field of computer systems performance evaluation would reap the benefits of many years of engineering experience.

6. REFERENCES

[1] Abrams, M. D., and Treu, S., A methodology for interactive computer service measurement. *Communications of the ACM* 20(12) (December 1977), pp. 936-944.

[2] Diday, Edwin, Problems of Clustering and Recent Advances, *Laboratoire de Recherche en Informatique et Automatique*, Research Report No. 337, January 1979.

[3] Ferrari, Domenico, *Computer Systems Performance Evaluation*, Prentice-Hall, 1978.

[4] Ferrari, Domenico, Characterizing a workload for the comparison of interactive services, *Proceedings 1979 NCC*, pp. 789-796. (a)

[5] Ferrari, Domenico, *Personal communication*, June, 1979. (b)

[6] Fogel, M., and Winograd, J., EINSTEIN: An Internal Driver in a Time-Sharing Envoronment. *Operating Systems Review 6(3)*, October 1974, pp. 6-14.

[7] Nolan, Lawrence E. and Strauss, Jon C., Workload characterization for timesharing system selection, *Software Practice and Experience 4(1974)*, pp. 25-39.

[8] Turner, R., Functional specification for SCRIPT-11. *Internal documents*, Digital Equipment Corporation, Maynard, MA, 1976.

[9] University of Michigan Computing Center, The D4.0 Terminal Simulator DSR and The D4.2 Terminal Simulator Monitor, *Internal Documents*, University of Michigan Computing Center, 1978.

APPENDICES

In the interest of paper conservation, the appendices have been omitted from this report. The complete set of appendices is available from the author, or they may be found in a M. S. Project by the same title filed in the E. R. L. Reference Library.