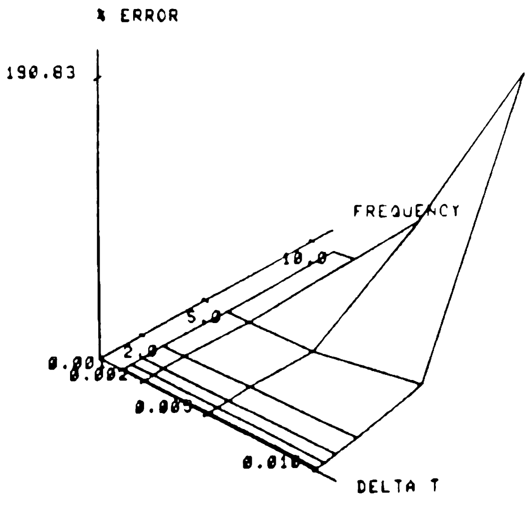

Figure 45. Third Order Taylor Integration Error, Damped Solutions.

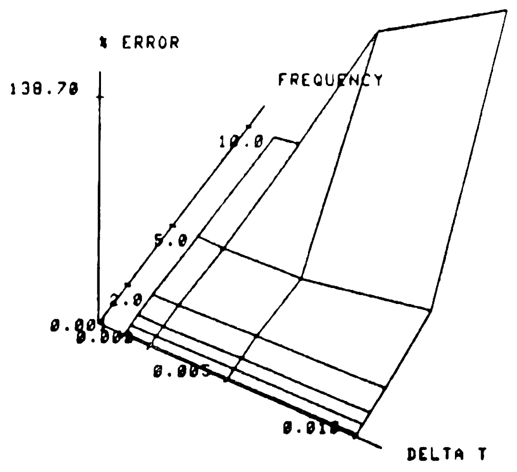

Figure 46. Fourth Order Taylor Integration Error, Damped Solutions.

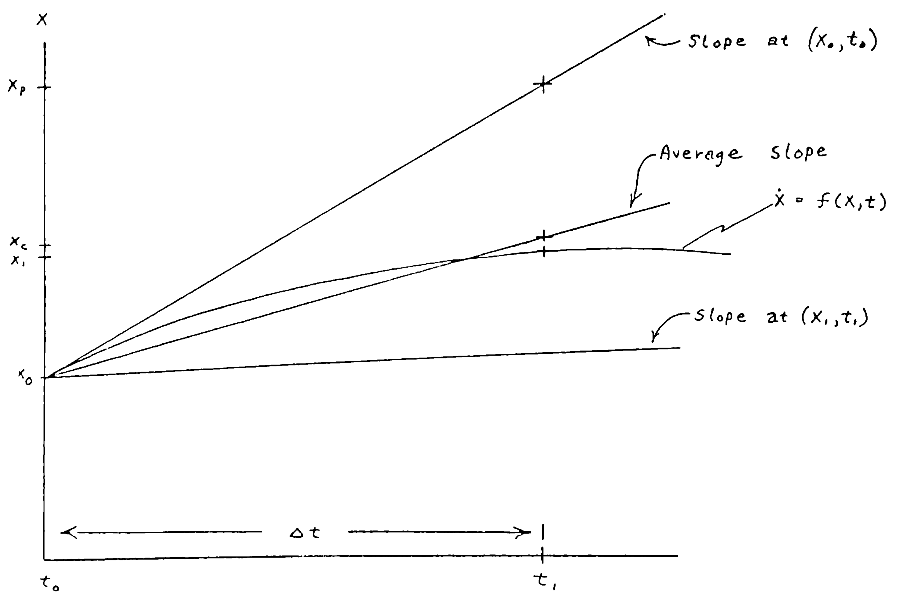

Figure 47. Corrected Euler Method

Software and Applications Modeling: An Example from Instructional Techniques

Abimbola Salako
Computer Science Department
University of Pittsburgh
Pittsburgh, Pennsylvania 15260

ABSTRACT

The need for understanding, characterizing, and modeling of application systems arises in performance studies, tuning of computer systems, and in the design of application responsive systems. However, very little progress has been made in the characterization and modeling of application systems. The work reported here is aimed at developing a model of Computer Assisted Instruction (CAI) application. We propose a functional model of instructional programs that describes different program types by a set of primitive building blocks representing application process states. Then, the apparent differences among the program types are explained in the light of this model. Starting with the primitive building blocks, it is shown how, by appropriate choice of parameter values, programs of varying descriptions can be generated. We also discuss procedures for the generation of synthetic programs (for simulation modeling) from the proposed model.

INTRODUCTION

The computing literature is replete with performance studies directed either at selection of equipment or at system tuning, while there have been very few reports in which performance is used as a basic framework for the design of a complete computing system. This may be attributed to the fact that most systems cater to a wide range of applications requiring differing and often conflicting criteria and levels of performance. However, when we consider systems that cater to a restricted class of applications such as a computer assisted instructional system (CAI) or an airline reservation system, a high level of performance can be achieved by designing the system in a bottom-up fashion rather than the conventional top-down approach. In the bottom-up approach, the software, operating system, and the hardware are custom designed to meet the demands and performance requirements of the application. Of course, the success of this approach will depend on how well the application area is understood and how accurately it can be characterized. The work reported here is the outgrowth of a simulation-based study of the design of a CAI system using the bottom-up approach. While the study was concerned with the design of the total system, the discussion in this paper is restricted to the modeling of the application system and the interested reader is directed to see Salako [8].

The characterization of the application is a necessary prerequisite for describing the workload in terms of the demands it imposes on the computing system. Furthermore, with a well defined and well characterized application, the task of generating synthetic workloads becomes trivial and the same application model can be used to develop a standard benchmark.

Before proceeding further, it is necessary to place this work in correct perspective with related work. Lucas [5] identified three objectives of performance study, namely: system selection, system modification or tuning, and system design. This work falls into the third category. While most systems are concerned with performance after implementation, this work tackles performance right from the design specification stage. The target system being a query-type system, the concept of workload can no longer be used in the restricted sense but must include the user (at the terminal console) whose rate of interaction affects the loading of the system. Thus, the requirements for workload characterization as outlined by Ferrari [3] apply to this broader definition of workload which we refer to here synonymously as the application system. One major distinction between this work and those involved with synthetic workload generation (e.g. Lucas [5], Sreenivasan and Kleinman [9], Oliver et al [7], and Strauss [10]) is that this model is based on a theoretical abstraction of the application as opposed to being developed through direct system monitoring or through the analysis of accounting data.

In the next section, we provide a cursory discussion of CAI application and while skipping the details of our analysis, we propose a formal model from which a functional model is derived. The parameterization of this model is then discussed in the following section, and the rest of the paper is devoted to a discussion of the procedures for parameter value assignment and generation of synthetic workloads.

THE APPLICATION SYSTEM

The CAI application system is composed of three subsystems, namely: the designer, the author, and the student. The designer component provides the tools for the author to convert his instructional specifications into computer programs, and those required by the student to learn his lessons. Software provided by the designer include:

1. the information management software which is responsible for collecting and analyzing student response and history records, and for maintaining the curriculum and lesson materials,

2. software for building, editing, compiling, testing, and executing lesson files, and

3. service routines for timing and terminal communication, and general software for managing system resources and interfacing hardware functions with software requirements.

The information management subsystem collects student performance and curriculum data necessary for monitoring student's progress, diagnosing his problems, and tailoring the lesson to suit his distinctive characteristics. There are three classes of student information files that must be built and maintained by the subsystem:

a. permanent data file containing student's background records, aptitude, intelligence, maturity, I.Q., interest and personality,

b. short-term history file for each subject area containing the records of student performances in related courses,

c. active data file containing each student's learning path, response latency, response log, success and error record, as well as terminal records.

The second class of data collected relates to the curriculum materials and teaching strategies. This information is employed in improving selection of curriculum material, course sequencing, and lesson presentation methodology. Typically, the information collected may be used to measure the effects of such instructional variables as: reinforcement, prompting, cueing, branching variables, question and response formats, and frame or step size. All the data collected must be reformatted, reduced, and analyzed to generate a summary to be used in decisions on subsequent programs or to generate reports which may be accessed by the lesson author, teacher, or curriculum designer.

The most important software package that must be provided is an author language with a compiler or interpreter for both on-line conversational mode and batch processing mode. This language must provide facilities for algorithmic computation, list processing, and simulation. It must also provide full answer processing facilities including timing, response pre-editing, syntactic analysis; answer matching such as: phonetic, numeric, algebraic, percentage, keyword, exact, and selected character string match; also, provisions must be made for unanticipated responses. Zinn [12] provides further details on required capabilities of instructional languages.

The designer subsystem represents the largest portion of the application system since the functions performed by the author and student subsystems use the services provided by the former. The function of the author subsystem involves building, editing, and updating of curriculum and lesson files as well as testing the lessons. The major files required for this function are curriculum files, course catalog, table of contents, index, and files for the actual lessons. On the other hand, the student subsystem's function is limited to the execution of the instructional programs.

Since the author and student subsystems define the pattern of requests for system services, the designer subsystem provides the tools for implementing these requests. Consequently, the workload characteristics are completely specified by the two lower subsystems which will henceforth be referred to as the application system. What characteristics does this application system possess? Is there a general structural pattern to these characteristics? The following paragraphs attempt to answer these questions.

CAI Programs

An analysis of existing instructional programs (e.g. Bitzer [1], Feingold [2], Meadow [6], and Wexler [11]) will show that there are three factors that account for their differences, namely: the method by which the instructional content presented to the student is obtained, the decision algorithm by which subsequent instructional steps are selected, and the level of interaction afforded the student by the system. Table 1 lists the alternatives available within each of these factors.

Table 1

Differences in Instructional Programs

1. Method of Obtaining Instructional Content:
   a) Selective
   b) Generative
2. Method of Controlling Instructional Sequence
   a) by student
   b) by system
3. Branching Modes
   a) Linear
   b) Non-linear
4. Response Formats
   a) Constructed
   b) Fixed format
5. Teaching Logics
   a) tutorial
   b) drill and practice
   c) gaming and simulation
   d) inquiry
   e) problem solving

In a selective program, all the instructional steps are pre-programmed and stored in a secondary storage while the generative program provides some complex algorithms for generating the instructional step presented to the student. The generative method thus demands a high degree of imaginativeness and generally places a heavy demand on the computing resources. In student controlled sequencing, the student determines what he wants presented to him as opposed to the system controlled case where the system determines what is best for the student. Given an instructional step in a linear program, there is only one possible successor step; while a non-linear program provides an array of alternatives at each step. The fixed response formats include YES/NO, TRUE/FALSE, and Multiple Choice and are generally easy to grade compared to the constructed responses which require extensive analysis. The instructional logics are self explanatory except for the inquiry logic in which the student learns by exploring a solution space through exchange of information with the system.

A Formal Model

Since the presentation of a detailed and logical development of the formal structure of instructional programs is beyond the scope of this paper, only the basic definitions will be given (for additional details, see Salako [8].

An instructional program, PROG is defined as the triplet:

$$\text{PROG} = \{S_u, U_o, M\}$$

where S_u is an ordered set of units,

$U_o \in S_u$ is the starting unit, and

M is the memory system.

A unit is defined as the pair, $\{S_F, F\}$ where S_F is an ordered set of frames and F is a decision function whose domain F^D is the set of parameters associated with members of S_F, and its range, F^R is a set of instructional units which are members of S_u and called the successor units.

A frame is defined as the primitive element of an instructional program consisting of the triplet, $\{C, P, a\}$

where C is the frame core,

P is a set of parameters, and

a is an activation pointer.

Finally, a lesson is defined as the quadruplet,

$$\{S, U_o, M, \dashrightarrow^*\}$$

where S is an ordered set of units such that,

$S \subset S_u$,

$U_o \varepsilon S$ is called the starting unit,

M is the memory system as defined in PROG, and

$\dashrightarrow^*$ is a transitive operator called the sequencing operator.

Programs are composed of an unordered set of units while lessons are composed of an ordered set of units which are selected by the sequencing operator (i.e. the sequencing controller which may be the system or the student). The memory system is an information net with a control and the information maintained can be grouped into three classes, namely:

i) instructional data base (curricula, course indexes, dictionaries, tables, and algorithms);

ii) instructional procedures (application packages and library routines, prototypes, generative routines, and general procedure); and

iii) student and instructional records (including permanent student history, student active records, and current performance records).

The motivation for the unit definition as an ordered collection of frames is the concept of a complete interaction on a single entity of an instructional material. By this, we mean the exchange of information between the student and the system until both are in mutual agreement that a new piece of material should be presented. The frame has been defined as the primitive constituent of an instructional program (contrary to the common usage of this term). The core is the program codes (like a subroutine or coroutine) with the activation pointer pointing to the entry or activation point. The frame parameters fall into three classes: local parameters (such as local variables in a subroutine), parameters for transmitting instructional information between the frame and the memory, and those for transmitting performance records. Parameters of the first type are called intrinsic parameters while the others are called extrinsic parameters.

The Unit Model

The approach taken in modeling the application system is to view the operations of its three components (the designer represented by the application software, the author represented by the curriculum material, and the student) as a single entity represented by a process. Through this approach, it is then possible to view the different process states as invocations of certain frame types, the invocation being effected by the sequencing operator. The definition of the frame core as a sequence of program codes is consistent with the process orientation since the dormant program codes would invoke the existence of the next process state upon being activated. Thus, the sequencing from frame to frame to define a unit is synonymous with transitions from process state to process state with a process state being defined by the application context and operations in the corresponding frame type. It must be emphasized that individual frames are of no significance to the process state except as identified by their type.

Figure 1 shows the states and state transitions required to define two unit types. States are indicated by S_j for j = 0,1,2,3,4,x,* and transitions between states are indicated by either solid arrows (system controlled sequencing) or broken arrows (student controlled sequencing). State S_x is the terminal state for a unit and from this state, branching must proceed to a new unit. S_* represents S_1, S_1', or the null state.

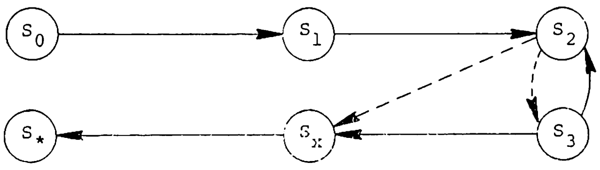

Figure 1.a Textual Unit State Map

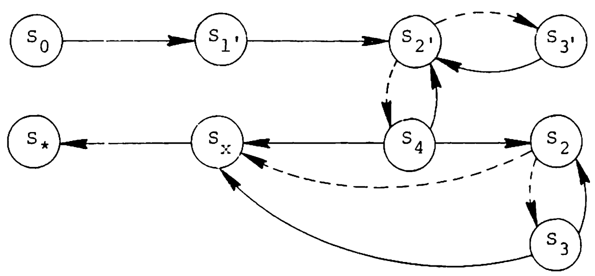

Figure 1.b Problem Unit State Map

Figure 1.a shows a Textual Unit in which the main frame, state S_1 depicts the presentation of a basic instructional concept or procedural instruction to the student. Following the presentation of the material, the student can either request additional information (S_3) or direct the system to advance to the next unit (S_x). Instead of the sequencing being controlled by the student, the system could control the sequencing as represented by the following state transitions: $S_1 \rightarrow S_2 \rightarrow S_x$. In this case, all the student need do is indicate when he is ready for the next material to be presented.

The Problem Unit, on the other hand, represents the presentation of a problem (S_1') which must be solved by the student or to which answer must be provided by the system before exiting from the unit. This problem may be as simple as a question requiring Yes-No type of answer or as complicated as developing or analyzing a model of a simulation or gaming problem. Following the presentation of the problem, the student can request additional information: $S_2' \leftrightarrow S_3'$; or propose a solution: $S_2' \rightarrow S_4$. Once the problem is solved, the student may ask questions of the system or direct the system to proceed to the next unit. Table 2 contains a description of the states (or frame types) in the units together with the representative processing functions performed in those states.

Table 2

Unit States and Representative Functions

Application System Initialization State (S_0):

. Loading of instructional data (course content, curricula, indexes, dictionaries)
. Loading of student records
. Loading of service routines and tables (input recognition routines, formalting and output display routines, library procedures, response matching routines, etc.)

Unit Initialization and Material Generation State (S_1, $S_{1'}$):

. Execution of content generation procedure
. Access to lesson files
. Initialization of measurement parameters (both intrisic and extrinsic parameters for student and instructional material)
. Output formalting

Response and Request Wait State (S_2, $S_{2'}$):

. Initialization of latency measures
. Initialization for response analysis and output display
. Data recording

Information Access State (S_3, $S_{3'}$):

. Input recognition and classification
. Search of tables, and various catalogs
. Output formalting and display

Response Analysis State (S_4):

same as above, and
. response processing using different matching algorithms
. system response generation (cues and hints, reinforements, etc)

Termination and Sequencing Decision State (S_x):

. Updating of student performance and history files
. Recording of instructional data
. Branching decision and selection of next unit

MODEL PARAMETERIZATION

Having developed the unit models above, the next question is, how can we use these models in a simulation environment? In particular, what characteristics of these models define the pattern (rates and distributions) of the computer system resources needed to perform the requisite instructional tasks?

These characteristics fall into three classes. The first is the implementation dependent characteristics which account for such factors as efficiency of coding, degree of code locality, program length, and ratio of I/O-related operations to pure computational operations. Due to the inexact nature of these factors and the lack of a standard methodology for measuring them, it is not possible to model them explicitly though they may be accounted for indirectly through some of the parameters of the simulation model. The remaining two characteristics are the student dependent and the instructional program dependent characteristics. The following two subsections will discuss these in more detail.

Student Variables

In our discussion of the unit model above, it was mentioned that state to state transitions can be effected by either the student or the system. The student controls this transition through his input at his console. The faster the rate of such input, the faster the change in state and therefore the heavier the demand on the system resources. Since in a typical system there would be several student consoles, an increase in the interaction rate from the consoles will impose heavier demand on the computational capability of the system. Thus student variables define the system workload (request rate and distributions). Student factors affecting the workload can be broken down into two classes: those affecting request arrival rate and those affecting the service rate (based on request type and their demand on system resources). Variables affecting the request rate are the student reading rate, typing rate, and think time.

Request Arrival Rate -- Quantitatively, the student reading rate can be estimated at about 260 words per minute, a figure usually given for the average adult reading rate. Considering that the typing speed of an average secretary is about 60 words per minute (or 300 char/min) an average typing speed of about 20 words per minute will adequately describe a student's typing rate. On the average, student think time will be less than ten seconds. Using these figures and the model reported by Fuchs and Jackson [4], it is possible to model, with high accuracy, the interaction rate for students at their consoles (i.e. the rate of transition from state to state in our model).

Request Type -- A student's aptitude, competence, background, interest, enthusiasm, and alertness will, in general, dictate the student's reaction to system actions. Thus, following a question, the student might give a correct answer, request the correct answer, request help or request additional information on the question. In general, there exists a set of alternative student actions at each system state. This is shown by the state transition matrix of Table 3. For each state S_j, the successor states to S_j are the matrix entries marked with an X or M along row S_j. Therefore, for each state S_j with more than one successor state, some type of probability-based switch must be used to select a successor state. Such a switch must be based on the profile of the sequence controlling student. To this end, we have defined a parameter called the student learning factor based on the composit profile of the student.

We thus define with each student context, a selection matrix with probability P_{ij} in matrix element, row i, column j such that

$$\sum_{j=o}^{\text{no. cols}} P_{ij} = 1 \qquad \ldots.(1)$$

$$P_{ij} = 0 \qquad \text{if corresponding entry in Table 3 is blank}$$

Furthermore, since loops exist in the state transition map indicating the existence of potentially infinite loops, the probability of staying within that loop must decrease as the number of times we have gone through the loop increases. In this case, the probabiltiy is modified by a factor derived from the learning factor and the loop count. Instances corresponding to this situation are shown by entries marked "M" in the transition matrix of Table 3.

Table 3.a

Textual Unit Transition Matrix

	S_0	S_1	S_2	S_3	S_x	$S_{1'}$	Null
S_0		X					
S_1			X				
S_2				M	M		
S_3			X		X		
S_x		X				X	X

Table 3.b

Problem Unit Transition Matrix

	S_0	$S_{1'}$	$S_{2'}$	S_2	S_3	$S_{3'}$	S_4	S_x	S_1	Null
S_0		X								
$S_{1'}$			X							
$S_{2'}$						M	M			
S_2					M			X		
S_3				X				X		
$S_{3'}$			X							
S_4			X	X				X		
S_x		X							X	X

Instructional Program Variables

While the student variables affect the rate of change of state as well as the state transitions (choice of successor state), the computational resources expended at each of the states is a function of the characteristics of the state. These characteristics are defined by the instructional variables described in Table 1. Table 4 describes in a rather qualitative way, the characteristics of the instructional variables that influence the basic resource demands. These resources are characterized in terms of main memory size, processor time, terminal input and output, and file storage input and output.

Table 4

Characteristics of Instructional Variables

I. Program Types
  A. Intrinsic Programs
    1. Shorter student performance records (or possibly none at all) recorded per unit.
    2. Simple algorithms required for implementing branching decision.
    3. Student records not used as branching parameter.
  B. Extrinsic Programs
    1. More detailed student performance records taken.
    2. Complex algorithms required for branching decision.
    3. Student records play an important role in branching decision.

II. Instructional Logic
  A. Tutorial
    1. High proliferation of textual type units.
    2. Mostly primary unit category with a few remedial and supplementary units.
    3. Low unit looping factor (non-repetitive program codes).
    4. Very lengthy output to student.
    5. Common instructional sequence consists of review of old material, introduction to new material, main tutorial sequence consisting of primary type units, practice sequence, and summary and conclusion.
  B. Drill and Practice Logic
    1. Primary unit type is the Problem unit with Textual units appearing mostly in supplementary and remedial materials.
    2. All unit categories may appear with equal probability with the student performance being the main controlling factor.
    3. Unit looping factor may be rather high especially in mathematically oriented subject areas and language drills.
    4. Length of output to the student is generally controlled by the question and response format employed.
    5. Typical instructional sequence is a main sequence followed by review.
  C. Games and Simulation
    1. The basic unit type is the Problem unit which actually represents a request for input parameters to the model being simulated.
    2. The prevalent unit category is the Primary unit while both Remedial and Supplementary units may be completely absent.
    3. Textual type units serve to describe the model and its parameters.
    4. Unit looping factor is generally high.
    5. Commonly, no student records are required for branching decision since branching is controlled by the parameter values supplied by the student as well as the pattern of student requests.
    6. Performance records may or may not be taken.
    7. Length of output to the student usually depends on the complexity of the model.
  D. Inquiry and Dialogue
    1. Branching or unit selection is jointly controlled by system and student but may be solely controlled by student.
    2. Common unit category is the supplementary unit.
    3. There is a high reference rate to course and curriculum index which may be core-resident or paged in from file memory.
    4. Units are generally not in a particular sequence since selection of units is often controlled by the student.
    5. When branching is done by the system, it is usually extrinsically controlled.
    6. This teaching logic requires a highly organized data base.

III. Question and Response Format
  A. Yes/No and True/False
    1. Simple procedure required for response processing.
    2. Length of question may be of the same order as that of constructed response-type question but much less that "multiple choice" type questions.
    3. Expected input length is generally of a fixed length.
  B. Multiple Choice
    1. Simple procedure required for response processing.
    2. Output length (question length)may be a factor of two to four times longer than Yes/No or constructed response-type questions.
    3. Input is generally of a fixed length.
  C. Constructed Response
    1. Very complex procedure required for response analysis.
    2. Length of input is generally variable though usually delimited by a given maximum value.

PARAMETER VALUE ASSIGNMENT

We established above that the application system is composed of the designer, the author, and the student subsystems. The designer subsystem is represented by the various programming software and service routines that must be accessed by the other two subsystems. The author subsystem provides the instructional programs and thus its effect on the application system is characterized by the instructional variables. The student subsystem, on the other hand, is modeled by parameters that describe his reactions at the student console. In order to generate the application system synthetically, we must assign values that reflect the resource demand represented by these parameters.

We have further indicated that the effects of designer variables are not directly modeled. Student variables mostly affect the demand rate and values have been given for the reading rate, typing or input rate, and the think time. Instances of student learning factor can be generated with a given mean and standard deviation to reflect different mixes of student population. What is left then, is to account for the effect of the instructional variables which we have shown to influence the basic resource requirements.

Since resource demands will of necessity be system dependent and any values assigned must be based on a specific instructional program implemented on a specific system (e.g. obtained through monitoring), the abstract or theoretical nature of this model precludes any meaningful value assignments. Therefore, the approach taken is to develop weighting factors representing the relative resource demands that would be experienced due to the effect of each of the parameters. These weights are obtained by viewing the resource demand levels as lying at discrete points or intervals of a continous spectrum. Through an analysis of each variable, it is then possible to place the resource demand corresponding to the variable along the spectrum. Using resource demand spectrum of lengths two and three in conjunction with the variables given in Table 4, the resource weight table of table 5 was generated. The table can be used as follows: assuming a mean value for a resource demand, then the equivalent resource demand for a unit with a particular characteristic is obtained by multiplying the mean by the wieght in the resource row entry corresponding to the column containing the variable.

Table 5

Resource Demand Ratio for Instructional Variables

	SELECTIVE PROGRAM	GENERATIVE PROGRAM	LINEAR PROGRAM	INTRINSIC PROGRAM	EXTRINSIC PROGRAM	CONSTRUCTED RESPONSE	YES/NO TRUE/FALSE	MULTIPLE CHOICE	TUTORIAL LOGIC	DRILL AND PRACTICE	INQUIRY & GAMES SIMULATION
PROGRAM SIZE	1	2	1	2	3	3	1	2	2	1	3
CPU TIME	1	2	1	2	3	3	1	2	2	1	3
TERMINAL INPUT SIZE	-	-	-	-	-	3	2	1	1	2	2
TERMINAL OUTPUT SIZE	-	-	-	-	-	1	1	2	2	1	2
FILE MEMORY INPUT	1	2	1	2	3	3	1	2	1	1	2
FILE MEMORY OUTPUT	1	2	1	2	3	2	1	1	1	1	2

CONCLUDING REMARKS

Through an analysis of instructional systems, we have developed an abstract model of the application system viewed as a composition of the designer subsystem represented by the application software; the author characterized by instructional variables; and the student, characterized by parameters that describe his terminal behavior. While the model is fairly parameterized, we feel that the parameterization could be further refined, though by our estimation, this would require a considerable amount of effort. A formidable problem in the parameterization process is that a software system structure is highly dependent on its production environment. This makes it almost impossible to parameterize a software system without identifying it with a specific system environment.

We are plagued by the same problem in our attempt to calibrate (or assign values to) the parameters of the model. Beyond the level of abstraction, there is nothing like a representative software and each software system is descriptive of the idiosyncracies of its developer. In an attempt to calibrate the model, we spent a lot of time trying to analyze three different instructional software implemented for different systems. The differences were so vast that the best we could derive from the exercise was the contents of table 5 which is far from being satisfactory; and in terms of our efforts, rather unrewarding.

From the model presented, it is still possible to generate synthetic workloads; however, the calibration of such a workload cannot be done independently of a specific system or a family of related systems. With this model, it is possible to describe synthetic modules for instructional systems similar to those described by Lucas [5] but more exact and detailed than the general modules he specified.

References:

1. Bitzer, D. L., and D. Skapadas, "The Design of an Economically Viable Large Scale Computer Based Educational System", Computer Based Educational Research Laboratory Report No. CERL X-5, University of Illinois, Urbana, February, 1969.

2. Feingold, S. L., and C. H. Frye, "User's Guide to PLANIT", System Development Corp., Santa Monica, 1966.

3. Ferrari, Domenico, "Workload Characterization and Selection in Computer Performance Measurements" IEEE Computer Magazine, July/August, 1972, pp. 18-24.

4. Fuchs, E., and P. E. Jackson, "Estimates of Distributions of Random Variables for Certain Computer Communications Traffic Models", Communications of the ACM, Vol. 13, No. 12, December 1970, pp. 752-757.

5. Lucas, Henry C., Jr., "Synthetic Program Specifications for Performance Evaluation", Proceedings of the ACM Annual Conference, August, 1972, pp. 1041-1056.

6. Meadow, C. T., D. W. Waugh, and E. E. Miller, "CG-1: A Course Generating Program for Computer Assisted Instruction", Proceedings of the 1968 ACM National Conference, pp. 99-110.

7. Oliver, Paul, et al, "An Experiment in the Use of Synthetic Programs for System Benchmarking", AFIPS Conference Proceedings, 1974 National Computer Conference and Exposition, Vol. 43, 1974, pp. 431-438.

8. Salako, Abimbola, "An Approach to the Total Design of Instructional Systems by Simulation", Proceedings of the ACM Annual Conference, August 1972, pp. 935-949.

9. Sreenivasan, K., and A. J. Kleinman, "On the Construction of a Representative Synthetic Workload", Communications of the ACM, Vol. 17, No. 3, March 1974, pp. 127-133.

10. Strauss, J. C., "A Benchmark Study", AFIPS Conference Proceedings, Vol. 41, Part II, 1972 Fall Joint Computer Conference, pp. 1225-1233.

11. Wexler, J. D., "A Teaching Program that Generates Simple Arithmetic Problems", International Journal of Man-Machine Studies, Vol. 2, No. 1, January 1970.

12. Zinn, K. L. A Comparative Study of Languages for Programming Interactive Use of Computers in Instruction, Final Report, ONR Contract No. 0014-68-C-0256, Center for Research on Learning and Teaching, University of Michigan, Ann Arbor, February 1969.