A STRUCTURAL APPROACH TO SIMULATION IN SUPPORT OF COMPUTER SYSTEM DESIGN

Melvin M. Cutler

Hughes Aircraft Company
Culver City, California

ABSTRACT

This paper describes the various features of a new approach to supporting computer system design by simulating the computer system as it progresses through levels of increasingly detailed design. These features include model specifications, use of structured programming, choice of simulation system or language, organization of simulation effort and personnel, the "chief modeler" concept, the model development process, modeler/designer information flow and interfaces, and the modeling of workload. An analysis is made of relative life cycle cost between the structured approach to simulation and the traditional approach of supporting a computer design effort.

INTRODUCTION

Simulation is all too often a stepchild of a computer system design effort. This characteristic is manifest in two ways:

1. The need for simulation is often not recognized until well into the design phase, resulting in a hastily written model which is error-prone, inflexible, and unstructured.

2. During the lifetime of a computer system design, the simulation effort is duplicated by as many as three or four mutually exclusive models -- an (early phase) user-oriented/problem-oriented simulation to define the data processing problem and to generate performance requirements, an (architecture phase) hardware/software model to make design tradeoffs and to predict performance, and an (implementation phase) detailed model which simulates program execution either at the machine instruction level or microinstruction level or both, to check out the hardware and software. The simulations are developed by different groups; they are discarded when no longer needed; and little or no information is passed from one modeling effort to another.

The overall simulation effort can be greatly improved by a unique method which might be termed "structured simulation" because of its analogy with structured programming. While structured programming refers to the top-down development of an eventually executable program, we define structured simulation as the top-down development of a single simulation program such that:

i. It begins when the design begins and is developed in tandem with the computer design at each phase of detail;

ii. At each phase of design detail, the simulation program is to function as the model used in that phase in a typical design effort as in 2., above.

STRUCTURED SIMULATION

A successful computer system design effort can be characterized by top-down development and stepwise refinement of algorithms and implementations; in short, structured design. The consequent simulation effort should also be structured; it should proceed as follows:

1. A modeler should be attached to the design team at the first opportunity. The modeler will develop the user-oriented/problem-oriented model of the system. This functional model is the top-level simulation of the system, and is preserved as it is refined during the design process.

2. When the design of the computer system itself is undertaken, the first model is refined so that the implementation of the functional model is reflected in hardware and software. The result is an architectural model of the system driven by real-world inputs.

3. When the computer system design has been fixed, the second model is refined to produce a simulator for program execution on the system at the instruction or microinstruction level. This third model is used for hardware and software debugging, and performance evaluation.

The simulation program which implements the model must use structured programming techniques so that it can be carried through these phases of a design effort. Adequate time must be devoted at the beginning of the project to the design of the

model, since the structure is preserved, whether it is good or bad. A critical part of model design is the choice of implementation language.

MODELING LANGUAGE

In the typical unstructured simulation effort described earlier, a basic incompatibility among the models developed for the design effort results from the different languages used to implement the simulations. The problem-oriented model is often written in a queuing or process-oriented language such as GPSS. The architecture-oriented model leans toward an event-oriented language like SIMSCRIPT. The debugging model is usually written in a scientific language such as PL/1 or FORTRAN, since the implementers of this model are normally unfamilar with simulation languages. It is clear that a structured simulation effort which refines a single simulation program into the three modeling levels must employ one language which is extremely flexible. There is little doubt that ECSS II (Extendible Computer System Simulator II) [1] is the appropriate language:

- ECSS is a high-level simulation language with powerful constructs which are ideally suited to a concise and a readable high-level computer system model;
- SIMSCRIPT, a lower-level simulation language with powerful scientific language features, is a sublanguage of ECSS;
- CSP II [2], a system for writing structured, table-driven computer system simulations, is written in SIMSCRIPT II.5 and thus can be incorporated into an ECSS-based structured simulation.

In summary, it would be difficult to justify using any language except ECSS II to implement three models which have, in the past, required three different languages. The flexibility of ECSS II is necessary to reduce the risk of the new approach, but it does not guarantee success. It is also important to reevaluate the model development process.

STRUCTURED MODEL DEVELOPMENT

It has become accepted that the use of advanced software technology is necessary to reduce the cost of a program through its lifetime. Thus, various techniques, lumped together as "structured programming," have been developed to meet this goal. A simulation program, despite its being a very specific type of software system, can benefit from the use of structured development techniques. Discussion of the use of these structured development techniques for simulation models is outside the scope of this paper; one comprehensive tutorial can be found in [3]. What will be emphasized here are the additional constraints on model development necessary to achieve the top-down refinement of a simulation model of a computer system such that at any phase of the computer system design the model meets the needs of the designers.

During its development, the model must perform the functions of the three or four models of the typical unstructured simulation effort described earlier. This can be accomplished by taking "snapshots" of the program as it passes through the desired levels of modeling detail. A "snapshot" of a simulation program is a partially developed version which may be used as a model for one of the three or four different levels. This process is illustrated by Figures 1 through 3, which show the progression of model development as well as the information required as input to the simulation.

FIGURE 1. LEVEL 1 PROBLEM-ORIENTED SYSTEM MODEL

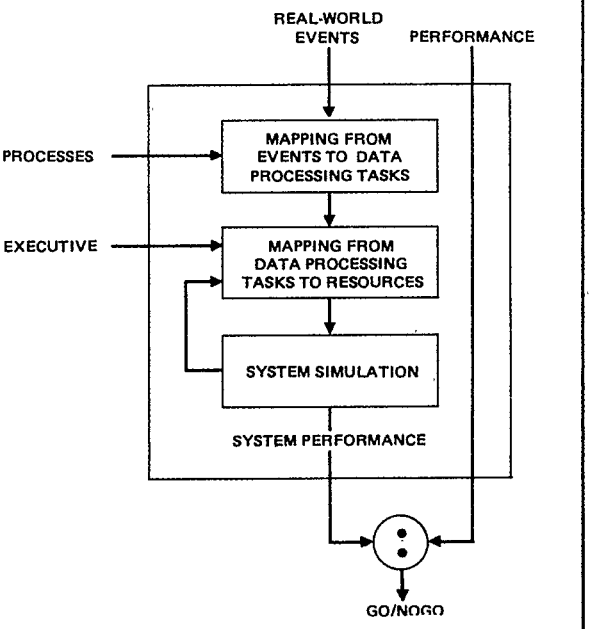


Figure 1, illustrating the level 1 problem-oriented system model, indicates the response of the simulated computer system to real-world inputs. At this level, a workload stimulus triggers the processing of tasks according to the executive program design and the available computer system resources. The functional sequence in Figure 1 begins with a table which maps a real-world event into one of a set of computer system processes. A process is a flow chart of data processing tasks which require resources such as CPUs, memories, transmission paths, and other devices. In ECSS, a process is defined by a JOB DESCRIPTION within the LOAD DESCRIPTION. The next box of Figure 1 indicates that a process becomes executable when the resources it requires become available and the executive chooses to schedule it; at this point the next pending task of this process is scheduled. ECSS has built-in facilities which simplify the simulation of the interaction of system resources with the tasks which utilize them. The lowest box of

Figure 1 stands for this process; the output of this function is the passage of time and the state changes of the system. If the system performance is within limits, the level 1 model has served this stage of the design.

A simple system illustrating a level 1 model might involve a real-world input of a signal representing an operator pushing a button at a particular time. This requires execution of a process which consists of a program to be run on a particular CPU, followed by a message sent over a particular path to an output device. The executive might simply give each of the two tasks of this process highest priority for the resources and the execution of each task on the system might be described by an ECSS EXECUTE statement and a SEND statement, respectively. The performance requirement might be a limit for the time delay between the pushing of the button and the completion of the output.

In Figure 2, the refinement of the level 1 system simulation model into an architecture-oriented computer timing simulation is indicated. At this level, the input data are data processing tasks mapped to resources, either using the level 1 mapping of real-world events or synthetic benchmark tasks. The latter may be necessary if performance requirements are stated in terms of instruction throughput for a mix which is independent of any particular system input. Because the level 2 expansion of the system simulation box of level 1 simulates only the timing of instructions, it must be provided with a mapping from data processing tasks to instruction streams. In the example used for level 1 above, the EXECUTE and SEND statements would be expanded to specify the sequence of instructions necessary to do each task, as noted in the top box of Figure 2, and each of these instructions would be expanded to specify the sequence of computer events which executes the instruction, as noted in the middle box of Figure 2. The first expansion, referred to as a dynamic characterization of applications programs, is a trace of an executed program provided by some "oracle," and is usually an educated guess. The expansion of each instruction type depends on the architecture of the functional units of the computer system. Because the basic executional entity of ECSS is the instruction, SIMSCRIPT is used to process expansions below the instruction level. A simple but illustrative architecture might map each instruction type into the following sequence of computer events: instruction fetch, instruction decoding, operand fetch, execution, storage of result, and calculation of next instruction location. The lowest box of Figure 2 indicates that system performance at this level is related to the time to schedule and perform these functional computer events, rather than to the computational results of these events.

This situation suggests that in the last level of refinement, illustrated in Figure 3, the simulation program and the simulated computer become indistinguishable. To bring this about, the mapping of data processing tasks to instruction streams is decomposed, instruction semantic information at the instruction or microinstruction level is added to the mapping from instruction streams to computer events, and execution simulation capability is added to the computer system timing simulation.

FIGURE 2. LEVEL 2 ARCHITECTURE-ORIENTED REFINEMENT OF SYSTEM MODEL

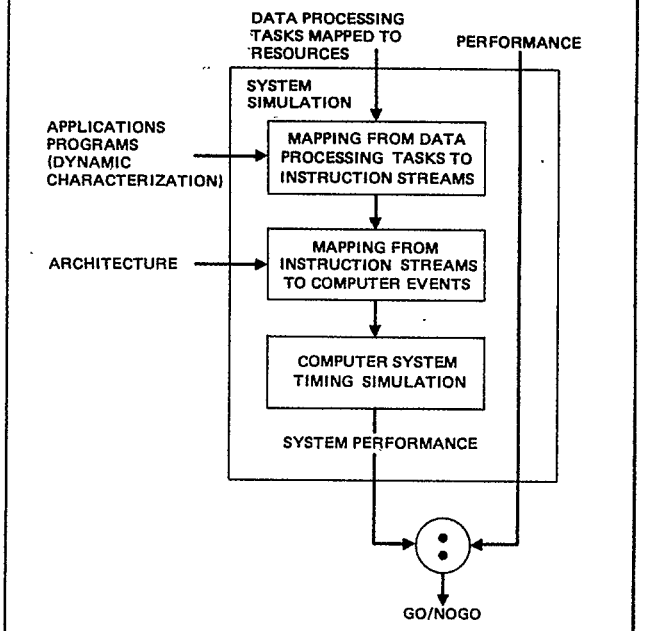
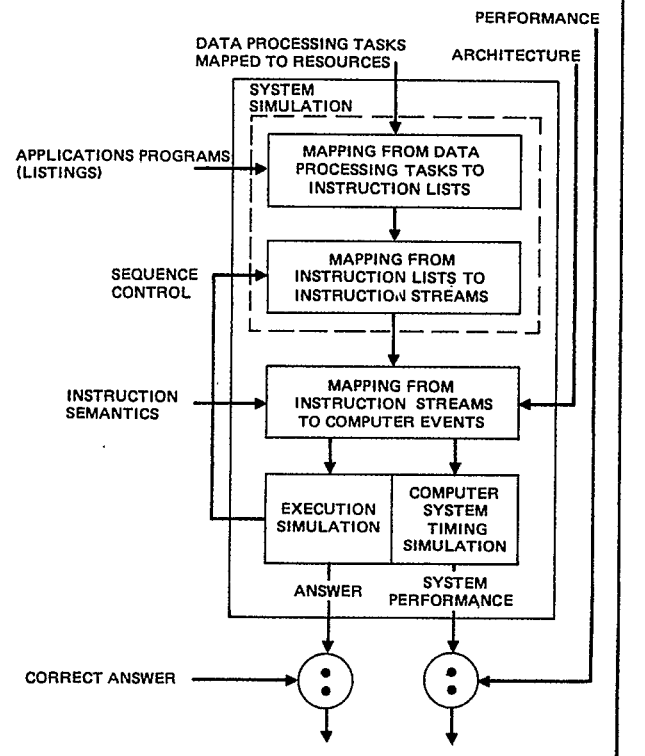


FIGURE 3. LEVEL 3 EXECUTION-ORIENTED REFINEMENT OF SYSTEM MODEL



The execution-oriented model performs the mapping from data processing tasks to instruction

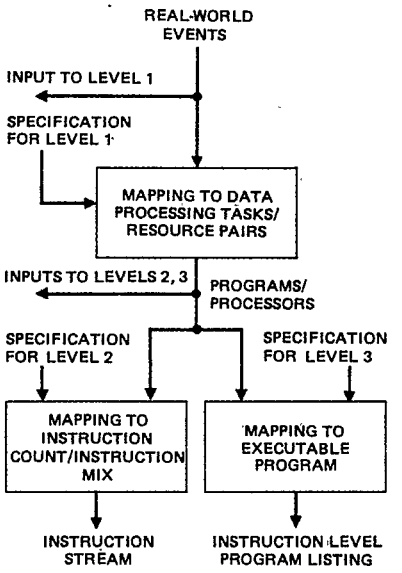
streams in two steps, as denoted by the dashes in Figure 3. The data processing tasks are presented to the simulation program in the form of listings of applications programs to be executed. These listings are converted to the dynamically characterized instruction traces of the level 2 model via the mapping of the second box of Figure 3; this mapping uses program sequence control information such as results of tests of variables from the execution simulation of the computer simulation. The mapping from instruction streams to computer events of the level 3 model is a refinement of the mapping of the level 2 model which introduces instruction semantic information for the execution simulation component of the computer simulation. For instance, an ADD and a SUBTRACT instruction are usually equivalent in a level 2 model, though they may have very different effects on the sequence control and answers of a program. Thus, Figure 3 indicates the provisions of the model for these new inputs and outputs. The bottom box also emits a new aspect of performance evaluation, the simulated answer, for comparison with the correct answer. This is the most we can expect a simulation to do.

WORKLOAD MODELING

It is apparent in Figures 1 through 3 that a significant effort is involved to refine the system workload in this structural approach as the computer system design progresses. In reality, the refinement of system workload specification parallels the software design process. Thus, the aspects of workload to be specified at each level of model development correspond to the natural progression of real-time software development. At level 1, the computational processes induced by real-world events are defined and are coordinated by assigning them to available resources via the executive program. At level 2, the system workload is characterized by refinement of the applications programs, which are elements of the level 1 processes, into sequences of instructions. These sequences are based on estimates of instruction mixes as represented by traces of programs; they are rarely given explicitly, but are generated using Monte Carlo methods. A level 2 simulation program also requires sufficient information about each instruction to be simulated in order to time it properly. For instance, a floating point addition execution often depends on the number of shifts necessary to equalize exponents; workload generation would require the association of a shift count with each floating point addition. Also, the level 2 model requires a refinement of the resource utilization of each workload component. While the level 1 resources of processors, memories, etc., were utilized by the workload components of jobs, messages, etc., the level 2 resources are computer functional units, as defined by the architecture, which are utilized by the level 2 workload component of instruction entities. Processing the workload is viewed as a series of computer events whose time is measured.

Level 3 requires no further workload refinement except to initialize the variables to be processed by the simulated system. The generation of the sequences of instructions which trace the execution of the simulated program is accomplished by the simulation program itself. Thus, the information which is a part of the workload specification of a level 2 model is integrated into the computer system simulation in a level 3 model. Using the floating point addition example, the shift delay to equalize exponents is determined by processing the actual operands and using the instruction semantic characterization to determine the delay. Figure 4 helps to illustrate the relationships among workload specifications for the three model levels; levels 2 and 3 are different refinements of level 1 workloads with approximately the same amount of detail.

FIGURE 4. WORKLOAD SPECIFICATION FOR MODEL LEVELS



To avoid haphazard development of workload levels it is necessary to structure the workload generation process along the lines of the mappings indicated in Figures 1 through 3. In this way, the same top-level workload used in the level 1 model can also be fed into the lower level models, providing workload consistency as well as model consistency from level to level. Another advantage of this structure is that the workload refinement for the simulation program can be implemented using a table-driven, automatic system; Figure 5 of [2] illustrates such a system for CSP II. Figure 4 is an idea of how a workload refinement system might go from level 1 to levels 2 and 3. At level 1, workload specification associates a process with a real-world input; the process is characterized by a flowchart of its tasks and the resources they utilize. This characterization is suitable as input to the levels 2 and 3 models, but it must be further refined to the instruction level to drive the computer simulation in both of these models. The mapping to refine the workload of a level 2 model might be a characterization of a task as a specific

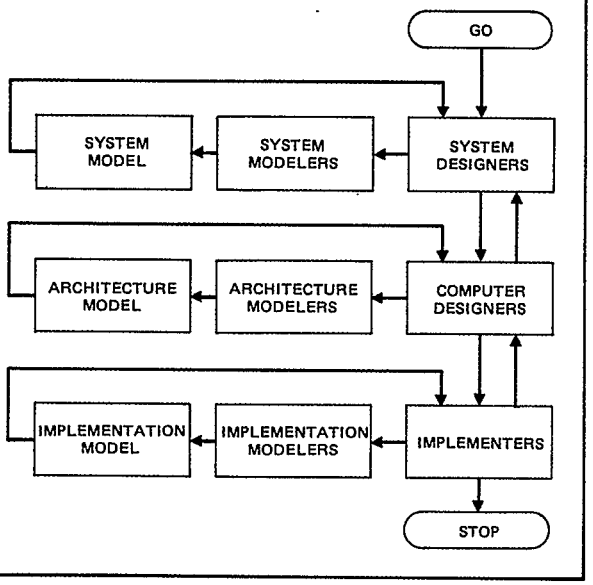
number of instructions of a certain mix, as in Figure 4. The workload of a level 3 model must be an executable program which halts after executing a certain number of instructions, given legal input.

While workload refinement is an important aspect of workload modeling, it is also necessary to design the workload format so that it need not be redesigned with changes in the computer system design or even with new design efforts, and so that it can be used at all levels. This problem was solved during the design of CSP II; thus, [2] should be consulted for further discussion.

ORGANIZING THE SIMULATION EFFORT

Though the importance of the proper structure for the simulation programs and their workloads should not be ignored, coordinating the generation of the model phase-by-phase presents difficult problems not solved by prior experience. The typical multi-model generation effort has flowed as in Figure 5.

FIGURE 5. TYPICAL MODELING EFFORT FLOWCHART

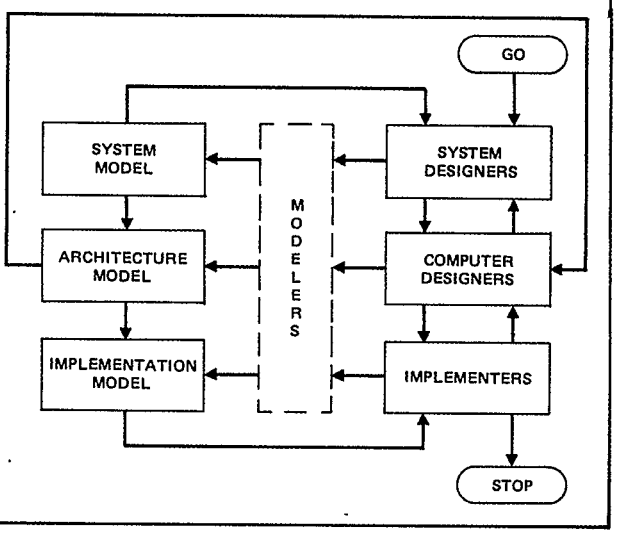


"Horizontal" information flow in this diagram is quite efficient, while "vertical" information flow is inefficient and unreliable when the model is in the loop. The structured simulation effort lends itself to a more effective organization, represented in Figure 6.

Horizontal and vertical information transfers are both facilitated using this structure. Flow among designer groups is as before. Flow among models can occur in two modes:

1. In the generation mode, the system model is first developed and saved (level 1). Next, it is refined into the architectural model and saved (level 2). Finally, the architectural model is refined into the implementation model and saved (level 3).

FIGURE 6. STRUCTURED MODELING EFFORT FLOWCHART



2. In the revision mode, the flow depends on which level requires revision. Revision at the system level requires the altered component to be modeled at all three levels using the structured approach as in the generation mode. An architectural revision not affecting the system model is implemented at levels 2 and 3 in a structured manner. Finally, implementation changes need only affect the implementation model, level 3.

Flow among modeling groups, in the structured effort, is implicit. The diagram pictures the modelers as a cohesive group, a departure from past, unstructured efforts. In the past, simulation was performed by system designers, computer architects, and hardware/software designers assigned to a modeling task. In a structured effort, all levels are implemented by a single modeling group able to converse with those three groups; this group need only be able to program in ECSS/SIMSCRIPT. The modeling group should be attached to the project organization – not divided among the three design suborganizations – and headed by a "chief modeler," analogous to the "chief programmer" of structured programming. The modeling group is allocated in the natural way during the two modes of model development (see above). Such an organization of the modeling effort will facilitate the blossoming of simulation as a key to a computer system design effort.

COST FACTORS

While there is little doubt that the nonrecurring programming cost of a structured simulation effort is less than for the traditional approach, it has been found that the level 3 simulation program of a high-speed computer system may possess run-time inefficiencies which limit its utility. Even a partially structured model development which refined a simulation from level 2 to level 3 incurred computer charges of approximately $.01 per instruction and was thus used only for

time-critical benchmark routines. One might therefore expect that a microinstruction level version of the model might be used even more sparingly. Of course, such experiences with simulations of computers with complex, highly parallel architectures can hardly be extrapolated to the current microcomputer generation. A microcomputer or a network of microcomputers, limited by advancing technology to a cost-effective life of only a few years, will likely present a positive case for the economical use of a three or four level structured model design. For other computer systems, the level 1 to level 2 refinement will be justifiable, while the lower level or levels may depend on the expected use of the models of that detail.

CONCLUSIONS

A new approach to supporting an entire computer system design effort by top-down development of a single simulation program has been described. Through experience in applying this approach, we will be able to evaluate its various components – project organization, simulation language, the modeling of workload, model levels and their development, and life cycle cost – and improve the baseline disclosed above. Through experience in observing the traditional approach, we are convinced the top-down method must eventually succeed.

REFERENCES

[1] Kosy, D. W., "The ECSS II Language for Simulating Computer Systems," RAND Corp. Report R1895-GSA, December 1975.

[2] Iwata, H. Y., and Cutler, M. M., "CSP II - A Universal Computer Architecture Simulation System for Performance Evaluation," 1975 Symposium on the Simulation of Computer Systems, pp. 196-206.

[3] Willis, R. R., "Structured Model Development Techniques," 1974 Symposium on the Simulation of Computer Systems, pp. 133-144.