Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

SOFTWARE CHARACTERIZATION INDEPENDENT OF HARDWARE

Lawrence L. Rose
Computer Science Department
University of Pittsburgh
322 Alumni Hall
Pittsburgh, PA 15260

ABSTRACT

This paper describes a software simulation methodology which is being developed for flexible, macro-level computer systems modeling. This methodology enables one to vary the host hardware configuration under consideration without requiring alterations to the software system characterization, or vice versa. The SOFT-SIM methodology forms the basis for a computer systems simulator which will exhibit independent, data-defined software and hardware specifications. Fully defined software systems and/or probabilistic job mix scenarios can be addressed with this methodology.

1. INTRODUCTION

Computer systems simulation is becoming an increasingly more difficult endeavor as software and hardware systems grow in complexity. This research emanates from efforts to solve the problem of hardware sizing for proposed software system designs. Given a detailed software specification, how does one determine the proper host hardware system, or choose among alternatives? Historically, vendors have provided benchmark data on various software functions such as searching, sorting, matrix inverting, etc. to give an indication of execution speeds and i/o requirements (Arbuckle 1965, Drummond 1969, Lucas 1971, Buzen 1978).

However, one can readily observe that a complex set of interconnected software routines in an on-line man-machine environment cannot be accurately assessed by modeling each software process in isolation. Furthermore, one cannot model software processes accurately without defining the specific hardware resources to be utilized by each process. The result is that most simulation models of large, complex hardware/software systems are complicated, unique, and ungeneralizable. This is due to the complexity of the hardware/ software definitions and to the specificity requirements that bind the software processes to the hardware resources.

One objective of this author's research over the past several years has been to reduce the complexity of producing accurate software characterizations for computer systems simulation. This has been achieved, at the macro level, through hierarchical event-oriented modeling techniques (Rose 1981, 1982, 1984). Recent research has focused upon a general solution to the software-hardware binding problem, the results of which are the domain of this paper.

It is clear that a simulation model comprised of separate software and hardware component definitions provides the desired flexibility of use for software or hardware systems planning. The rub is that it is very difficult to define software processes at a detailed level without knowledge of the host hardware configuration. If this knowledge is explicitly used in the software definition, then it must be altered every time a different hardware configuration is considered. Given non-identical software definitions, it is difficult to prove that the same workload is processed by the different models.

This problem grows in complexity when the hardware configuration is a network of cpu's and devices. The methodology described herein provides an approach to properly binding independent software and hardware definitions. This enables either definition to change without requiring any alterations whatsoever to the other simulation component, be it hardware or software. The focus of this paper is on software specification, per se, and its binding to the hardware configuration.

2. SOFTWARE PROCESS MODELING

Software systems can be considered to be comprised of a network of low-level software functions (eg. job steps), each characterized at the macro level by an input, compute, and output cycle as shown in Figure 1 (Russ 1979). The software function resources include: an Input Device, a Cpu, Memory, and an Output Device. For full flexibility we consider executing on a network; therefore we must model

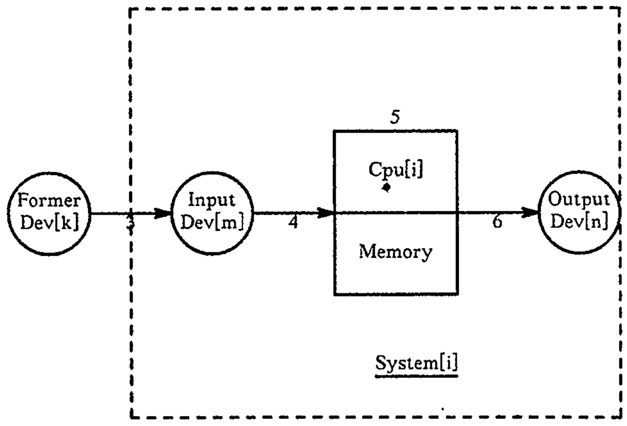

Figure 1. Software Function Simulation

the data transfer required if an input dataset resides on a device unattached to the host cpu.

This environment provides the flexibility requisite for hardware-independent software characterization. We let *swcode* be a unique encoding for each different software function/job in the host software system. Methodologically, we have the following sequence to follow in order to simulate software execution at the macro level (ie, no looping between steps 4-6):

1. **[System Determination]**Cpu i must be chosen based upon the software residency preferences of *swcode* and the residency of its input data set on Device k.
2. **[Device Allocation]**.Given the choice of Cpu i, Devices m and n (that are attached to this cpu) must be selected and allocated for i/o.
3. **[Data Transfer]**If m =/ k then the input data set must be transferred from device k to device m.
4. **[Input]**Input is simulated for function *swcode* ; transfer amount and channel attributes for connection and current usage are used to determine the time required.
5. **[Compute]**Cpu execution is simulated; the cycle requirements of *swcode* and hardware attributes of Cpu i (cycle time and current usage) are used to determine the time required.
6. **[Output]**Output is simulated, in a manner similar to step 4 above.
7. **[Device Release]**Devices m and n are deallocated as requested by the modeler.

Given this macro-simulator for software functions, a solution is required that enables one to fully define a software function without being overly specific in the software definition. First, the i/o devices cannot be specified below the generic class level(disk, tape, etc.) while still retaining hardware definitional independence. Second, each software function must be defined independent of the execution configuration, with the i/o device allocation and deallocation dynamically simulated to provide device and channel capacity/utilization statistics.

Our solution is to define SW as the generic software simulation function. It is invoked with the following arguments:

```
SW(swcode,indev,outdev,disp)

where:
    swcode = unique software function code

    indev  = generic input device              (1)

    outdev = generic output device

    disp   = device deallocation disposition .
```

Note that no mention of a specific cpu is made, nor are specific devices referenced. Yet SW provides the modeler the capability to invoke software with specific executional attributes (indexed by *swcode*), to allocate and utilize a specific class of input and output devices, and to deallocate devices at job termination. Thus the SW construct enables one to prescribe software functions in a detailed, yet hardware-independent manner. The following section details the additional constructs necessary during simulation to provide hardware binding for proper modeling of software execution.

3. SIMULATION-TIME BINDING

Generic software invocations such as those characterizable by use of the SW construct cannot be simulated without further support from the model in two areas: hardware assignment and hardware usage. For example, the call SW(SORT5,TAPE,DISK,NEW) requires the model to assign a cpu, mag tape, and disk to carry out the software function encoded as SORT5. After selecting this subset of the defined hardware system, the model must know something of SORT5's i/o and cp requirements so that hardware usage can be properly simulated.

These problems are solved by utilizing two additional data structures whose contents are defined via model inputs. The mapping FNCPU is user-defined to assign any subset of the set of modeled cpu's to each software function referenced by the model. Thus if we have:

$$\text{FNCPU(swcode)} = \quad d_1\ d_2 \ldots d_n \qquad (2)$$

then the software function *swcode* is restricted to the cpu's d[1], d[2], ..d[n]. Furthermore, Cpu d[1] is preferred (if available) since it appears first in the FNCPU mapping for this *swcode*. For every SW invocation, there is an implicit guarantee by the modeler that *indev* and *outdev* are attached to each of the cpu's in the FNCPU mapping for this *swcode.*

Once a cpu subsystem in the network has been chosen (using the above mapping) the hardware network must be searched to find devices of type *indev* and *outdev* for specific allocation requests to follow. In this way, we can make an unambiguous (yet dynamic) transformation from a generic software invocation to specific device utilization in a manner controlled by the modeler with independent model input describing the hardware network and the FNCPU mapping.

The second data structure requisite for simulating each of n software functions is the software descriptor array:

$$\text{SWDESC}(1{:}n,1{:}8). \qquad (3)$$

The first dimension of SWDESC is indexed by the integer code associated with each unique function name: *swcode.* The eight function descriptors include: i/o allocation sizes, i/o amounts, and cp cycle requirements (A+Bx+Cx**2+Dlog[x]) where x is the size of the data set invoking the software function.

The end result is that SWDESC, in conjunction with the variable *swcode* of the SW invocation transforms a job request into byte amounts for i/o timing and cp cycles for cpu execution timing. Other data, defined by input characteristics for the host hardware network being modeled, is utilized to determine the effective i/o and cpu time during the simulation. Thus the additional user-defined data structures FNCPU and SWDESC provide adequate binding during execution to simulate actual software functions on specific hardware devices.

4. INITIAL CONCERNS

The methodology outlined in the previous two sections suggests that one can successfully imbed a great degree of software/hardware independence into a software systems simulator. In fact, the CSAR simulator (Computer Systems AnalyzeR) developed at Battelle-Columbus Laboratories reflected much of this methodology (Rose et al 1982a,b). It was applied with reasonable success to the specific problem domain of dedicated software systems sizing. However shortcomings were encountered using this simulator that could not be resolved without major changes to the underlying methodology and the resultant implementation. Let us revisit Figure 1 for a critical look at the software characterization methodology and our initial implementation thereof.

The major methodological drawback was the requirement that each software job be characterized by a single i/o pair; generalization to n logical inputs and m logical outputs brings us closer to reality. This implies that a job should be triggered by the arrival of all of its inputs rather than invoked by the successful completion of its software predecessor. This further generalizes the software job structure to a fully connected network, with multiple predecessors and successors possible for each job.

The major implementation drawback was the requirement that the modeler definition of the software job network be imbedded in the simulation code. This required coding by a modeling specialist with extensive FORTRAN and GASP expertise. A more generalized implementation would be language-independent to enable the modeler to directly implement the software definition.

A lessor drawback lies in the SWDESC definition. Its cpu characterization of the algorithm order is in fixed terms of $A+Bx+Cx^{**}2+D\log[x]$, where x is the size of the input. A more general approach would input any algebraic expression of this definition.

Finally the GASP language has been incorporated into SLAM and is no longer supported; the latter is less oriented to user systems programming. Further research will utilize the process- oriented view as exemplified by SIMULA (Franta 1977) for computer systems modeling, in view of the above implementation drawbacks experienced with the earlier event-oriented effort.

5. SOFT-SIM PROCESS CHARACTERIZATION

Given the above mentioned concerns, a new methodology has been developed to solve the problem of modeling software processes independent of hardware. This methodology is named SOFT-SIM , for software simulation, and is in the process of being incorporated into a process-oriented data-driven computer systems simulator (Rose 1986). Figure 2 illustrates the more general view of a software job taken by this methodology. Based upon the modeler-defined inputs that characterize each job, SOFT-SIM derives the resultant network of software processes and carries out a process-oriented simulation on the modeler-defined hardware configuration.

In contrast to Figure 1, one observes that this new approach characterizes each software process with multiple (rather than singular) i/o datasets. The i/o logical dataset

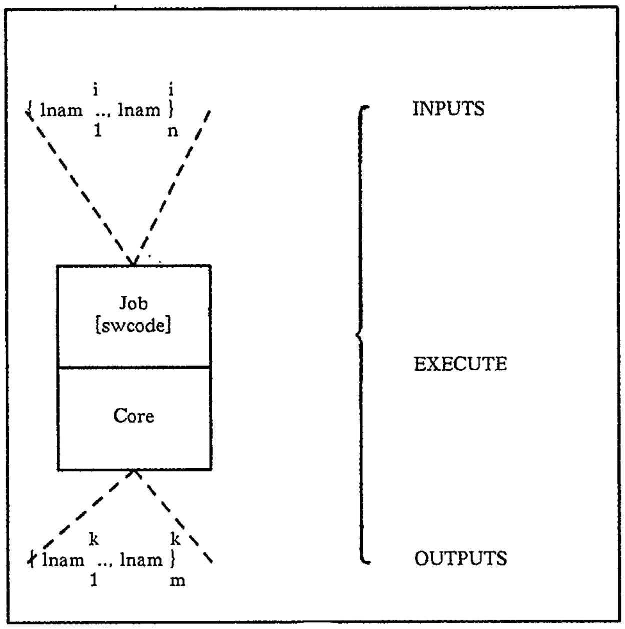

Figure 2. SOFT-SIM Job Characterization

names *lnam* are unique in that they may appear at most once as an input and at most once as an output. They serve to link together the software process network. The solution to process successor(s) is provided by the unique process whose input dataset name equals the output dataset name. An input dataset with no matching output dataset can be considered exogenous; an output dataset with no matching input dataset can be considered terminal.

The modeler thus defines the software system for SOFT-SIM as an independent set of software jobs (processes), each having a unique *swcode* and associated sets of logical input and logical output datasets. Each software process is data-defined, independent of other processes. This specification technique is sufficient to describe any hierarchical/networked software system of arbitrary heterogeneous complexity. This power is derived from the ability to hierarchically define the jobs to whatever level of detail is desired.

6. HIERARCHICAL JOB DECOMPOSITION

The motivation for hierarchical decomposition stems not only from detail requirements, but also from the dynamic nature of model specification in general. One does not know enough about the system or its problems at the onset to define it completely or determine the proper level of detail. By moving the software specification from the *sourcecode* medium to the *data* medium, we provide the flexibility necessary to respond to the dynamics of model construction and usage. Changes can be made to the software specification without endangering the model software validity.

Hierarchical decomposition or step-wise refinement of the software characterization can be accomplished by simply redefining a previously defined job. This enables an expansion of job detail such as shown in the following

Figure 3. (Note that the exogenous i/o sets must remain unchanged for both levels of definition.) This provides full modeling flexibility for carrying out simulation experiments using SOFT-SIM: easy updates of the model definition with no source code modifications:

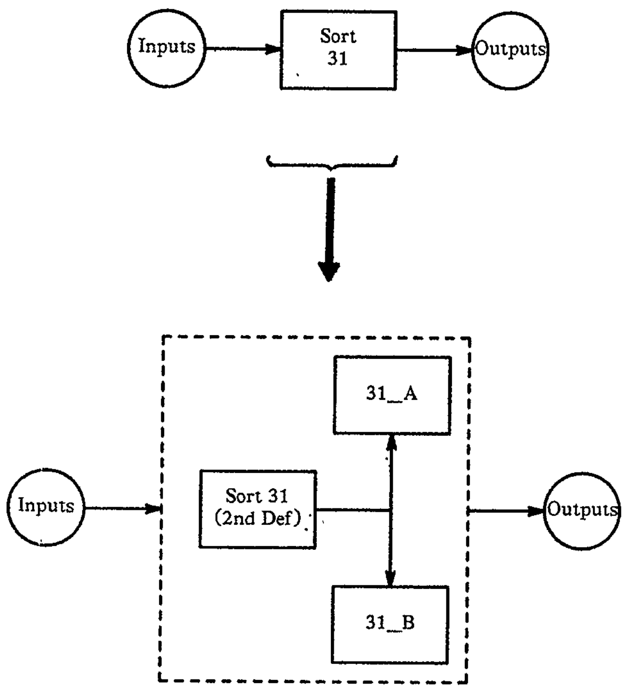

Figure 3. Hierarchical Job Definition Expansion

The updates are easy because we do not remove the former job definition, we simply add a later redefinition. For example, in Figure 3 we observe that Sort 31 has been expanded to three jobs: Sort 31(2nd def) followed by 31_A and 31_B. This is acomplished in the data definition by naming Sort 31(2nd def) outputs to be different from the original Sort 31 outputs, and sending some to 31_A and others to 31_B. The combined outputs of 31_A and B must be identical to the original outputs of Sort 31.

The fact remains that one could remove the entire original definition of Sort 31 and replace it with the new definition. It's just that this sort of destructive update is more prone to error, and it destroys the path of software redefinitions. This path serves as a historical marker; furthermore it need not be revalidated whereas the replacement (as opposed to the redefinition) software definition may include errors at higher levels of definition.

The second sort definition illustrated in Figure 3 would appear later in the input stream, and would be presumed to be the start of a lower breakdown of the software job Sort 31. Statistics are provided only for the lowest-level definition of a job. In this case, SOFT-SIM would derive statistics for the network of processes: Sort 31(2nd Def), 31_A, and 31_B. Similarly, any of these three jobs could be redefined at a lower level as detail so required.

Using this software process network scheme, one can define hierarchical networks at heterogeneous levels of detail. One can model as many job streams as desired. Probabilistic job mixes can be viewed as a set of independent software processes, with no successor/predecessor relationships. The arrival rates of the exogenous input datasets can be varied to provide the job mix desired.

This extension to multiple i/o datasets for software job characterization necessitated changes in the underlying modeling tenets of CSAR. In fact, the event-orientation of our earlier models was scrapped in favor of the more general (and hierarchical) process-orientation. The final section outlines the design for SOFT-SIM process modeling.

7. SOFT-SIM PROCESS SIMULATION

Given this more general data-defined process characterization, the SOFT-SIM methodology must base process invocation upon an extended SW construct which will be derived from the input rather than explicitly invoked from within modeler-defined source code. We can utilize the SW construct of Definition(1) by interpreting the arguments *indev* and *outdev* as pointers to the modeler-defined sets of *lnam* objects. The FNCPU mapping of Definition(2) remains unchanged, while the SWDESC descriptor matrix of Definition(3) must also be expanded to pointers for the multiple i/o attributes and the cpu cycle expression.

A new data structure is required to bind the logical dataset names within the modeler software definition to physical device references during simulation:

$$\text{DSGEN(lnam)} = \text{generic device.} \qquad (4)$$

This data structure has important ramifications, for it shows that SOFT-SIM invokes software processes based upon *lnam* arrivals rather than the completion of a unique *swcode* software process predecessor. The invocation of software processes is handled implicitly in SOFT-SIM (based upon the modeler-defined software network definition) whereas CSAR explicitly invoked processes by sequentially executing modeler-defined source code using calls to the SW function.

SOFT-SIM initiates model simulation by analyzing the *swcode* network definition, and triggering initial arrivals of all exogenous *lnam* datasets. Any *swcode* whose inputs have all arrived is immediately invoked using the expanded SW construct. At *swcode* completion each of the *lnam* outputs is routed (if non-terminal) to its successor *swcode* and the procedure continues until halted under modeler control.

This methodology forms the basis for the development of a SOFT-SIM simulator. Initial research has been carried out, using the MODULA-2 language for testbed purposes. This language has the necessary vitals for process-oriented modeling using abstract structures and modular software. It will enable the software realization to be developed hierarchically.

8. CONCLUSIONS

This new methodology for software systems simulation fully addresses the shortcomings of our earlier research effort. Of primary import is the fact that the requirement for a modeler defined *sourcecode* software definition was relaxed to a language and simulation-independent *data* definition. Further, this new definition allows far more complex software networks to be defined, and they can be easily refined by the modeler during the duration of the simulation exercise.

The objective of this research was to formulate a general methodology for characterizing and modeling software processes independent of hardware. The solution was developed herein and is being implemented in the SOFT-SIM model. Extending to a data-driven multiple i/o software-hardware hierarchy while maintaining the necessary bindings provides full modeler flexibility. The integrity of model results is enhanced by the top-down structure of the data-defined software hierarchy.

REFERENCES

Arbuckle, R. (1965). Computer Analysis and Thruput Evaluation. *NCC'65 Proceedings*, pp. 66-75.

Buzen, J.P. et al (1978). BEST/1 - Design of a Tool for Computer System Capacity Planning. *NCC'78 Proceedings*, pp. 447-455.

Drummond, M.E. (1969). A Perspective on Systems Performance Evaluation. *IBM Systems Journal* **8**(4), pp.252-263.

Franta, W.R. (1977). *The Process View of Simulation*, Elsevie, North Holland.

Kiviat, P.J. et al (1973). *Simscript II.5 Programming Language*, CACI, Arlington, VA.

Lucas, H.C.Jr. (1971). Performance Evaluation and Monitoring. *ACM Computing Surveys* **3**(3), pp.79-92.

Rose, Lawrence L. (1981). Hierarchical Modeling in GASP. *Proceedings of the Fourteenth Annual Simulation Symposium*, Tampa, FL, pp. 199-213.

Rose, Lawrence L. (1982). TRM: A Resource Model for Networked Processes. *Proceedings of the Fifteenth Annual Simulation Symposium*, Tampa, FL, pp. 211-221.

Rose, Lawrence L. and Freuler, F. Theodore (1982). A CPU Model for Concurrent Processing . *Modeling and Simulation* **13**(2), pp. 705-710.

Rose, L.L., Freuler, F. T., and Hochstettler, W.H. (1982). The TES/EMPS-IFE Simulation System. *Battelle Report*, Battelle Columbus Laboratories, 55 pages.

Rose, Lawrence L. (1984). A Hierarchical Multi-Level Interactive Systems Simulator. *Modeling and Simulation* **15**(5), pp. 2011-2018.

Rose, Lawrence L. (1986). The Development of HP-SIM: Hierarchical Process-Oriented Simulation Software. *Technical Report* **86**(1) , University of Pittsburgh, 19 pages.

Russ, Teodor (1979). *Data Structures and Operating Systems* , John Wiley & Sons, New York, NY.

AUTHOR'S BIOGRAPHY

LAWRENCE L. ROSE is an associate professor in the Computer Science Department at the University of Pittsburgh. He received a B.A. in mathematics from VMI in 1965, and the M.S. and Ph.D. degrees in computer science from Penn State University in 1967 and 1973 respectively. He was involved in personnel simulation modeling studies while with the U.S. Army (1967-1969). He was an assistant professor at SUNY-Binghamton (1973-1975) and Ohio State University (1975-1979). From 1979 to 1984 he managed the systems simulation group at Battelle-Columbus Laboratories. His current interests include simulation language development, simulation software engineering, and modeling computer systems. He is a member of ACM, SCS, and SIGSIM.

Lawrence L. Rose
Computer Science Department
University of Pittsburgh
322 Alumni Hall
Pittsburgh, PA 15260
(412) 624-6475 x3343