Proceedings of the 1983
Winter Simulation Conference
S. Roberts, J. Banks, B. Schmeiser (eds.)

REPRESENTATION AND MODELING OF DISTRIBUTED COMPUTER SYSTEMS

Armen Gabrielian
Hughes Aircraft Company
P.O. Box 3310
Fullerton, CA 92634

The general problem of the representation and performance-oriented modeling of the behavior of distributed computer systems is discussed. A description of an integrated System Design/Static and Dynamic Modeling (SD/SDM) tool set is presented. In SD/SDM, a data base organization is used for the representation of the architecture, load, application software and the operating system of a distributed system. Modeling is performed statically or dynamically through a generic simulation model that does not require recompilation. Applications of SD/SDM in actual projects are discussed briefly.

1. INTRODUCTION

A distributed computer system and data communication interfaces are common components of most current and expected future command and control systems. The design of such a combined computer and communication system is a complex process that begins with system requirements, proceeds through systems analysis and preliminary design, and progresses through performance modeling to final design definition. As systems are becoming more complex, system design consumes steadily increasing manpower, computer resources and elapsed time, jeopardizing the likelihood of producing a proper design. The magnitude of the associated effort requires that an integrated set of automated tools be utilized at all stages of design and development.

In the requirements definition area, there exist a variety of tools such as Problem Statement Language (PSL) from the University of Michigan, Requirements Specification Language (RSL) at TRW, Specification and Description Language (SDL) defined by CCITT and Requirements Language Processor (RLP) at GTE Laboratories (Davis 1982). Specialized languages such as SPECIAL at SRI international (Levitt et al 1979) and GYPSY (Ambler et al 1977) have been used to specify operating systems. None of these, however, interfaces conveniently with a comprehensive performance evaluation tool.

In the modeling area, queueing and simulation are the principal approaches. Queueing packages such as BEST/1 have been used successfully in a number of commercial data centers. However, queueing models are inflexible and there are strong limitations on the kinds of systems that they can model. Simulation tools, on the other hand, have the benefit of generality but are often costly and difficult to use. General purpose simulation languages require extensive programming and specialized languages such as ECSS II (Kosy 1975) share common drawbacks of most simulation tools. These drawbacks include the need for recompilation whenever a change in a design occurs and the difficulty of analyzing and verifying a system definition independent of the simulation context. An additional problem is that most simulation tools provide no aids to the design process itself.

In order to circumvent these problems, an integrated set of tools called SD/SDM (System Design/Static and Dynamic Modeling) has been under development at Hughes since 1981. SD/SDM provides a data base organization for system definition and automatic means of modeling the system behavior from the system definition. Two kinds of performance modeling are accommodated in SD/SDM: static analysis and dynamic simulation. Static analysis characterizes the load in each device, ignoring queueing delays, system overheads and system capacities. It is usually performed in the early part of system design. Dynamic simulation models the detailed time-oriented behavior of the system and provides

information about response times, operating system overheads, queueing delays and processor utilization.

SD/SDM is intended for use by systems engineering personnel and, to a large extent, eliminates the need for writing simulation programs. It has been used in a number of major projects to model the performance of systems containing up to 100 computers. Because of its ease of use and system-oriented framework, it has reduced the effort and time required to develop a model definition by an order of magnitude compared to some previous tools. In one specific case, the entire system definition and modeling process (static and dynamic) for a 60 computer configuration was performed within the period of one week.

A description of an earlier version of SD/SDM appeared in (Gabrielian 1982). Section 2 of this paper presents an overview of the current version of SD/SDM which is significantly more powerful. Section 3 discusses objects, processes and parameters and the manner in which they are used to describe a software architecture. Section 4 defines the approach to the representation and modeling of operating systems. Finally, Section 5 briefly reviews some of the applications of SD/SDM and discusses present plans for its further development.

2. OVERVIEW OF SD/SDM

The structure of SD/SDM is depicted in Figure 1.

SD/SDM contains three major components:

- System Design Data Base (SDDB)
- Static Analysis Model (SAM)
- Dynamic Simulation Model (DSM)

SDDB is a relational data base implemented in the very-high-level language SAS (Statistical Analysis System). Its structure is similar to the data bases generated through SDL (Simulation Data Language) as described in (Standridge and Pritsker 1982), but its use is somewhat different. SDDB is the repository of all information about the hardware, software and the load of a system. There are currently 12 relations or tables in SDDB, each of which is concerned with a particular aspect of the system. For example, the devices are defined in one table, the architecture in another, and the software and related data are described in the remaining relations.

The use of the SDDB concept provides several important advantages. It provides a convenient formalism for system definition that can serve as a communication medium among system engineers. It can be used to build up a system design library, allowing components of previous systems to be easily integrated into new designs. SDDB is also convenient for verifying the accuracy of a design specification prior to simulation. This is accomplished through various configuration and consistency/completeness reports.

The Static Analysis Model (SAM), which is also implemented in SAS, computes the processing requirements and the data traffic load of the system, ignoring queueing delays, system overheads, and resource capacities. It is generally used in the early phases of design to determine estimates of quantities of computers and peripherals and types of buses required. SAM is also used to determine the initial allocation of functions and data objects.

The Dynamic Simulation Model (DSM) component of SD/SDM is a simulation program that models the time-oriented behavior of a distributed computer system. DSM is implemented in SIMSCIPT II.5 and, because of its generic nature, does not have to be recompiled for simulating alternate architectures. DSM accepts as input an "interface file" that is optionally created by the SAM components of SD/SDM. The interface file is a "compilation" of the SDDB definition of the system and its load in a format that can readily be used for simulation. DSM produces a variety of reports on processor utilization, bus activity, queueing delays and response times.

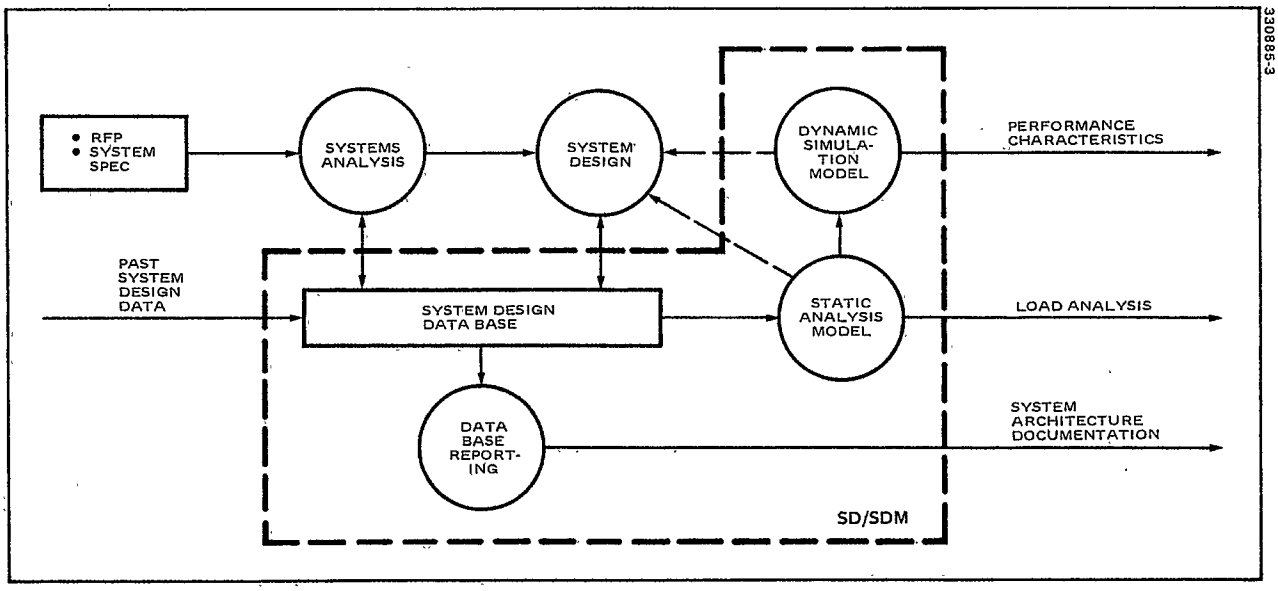

Figure 1. The Structure of the SD/SDM Tool Set

3. OBJECTS, PROCESSES AND PARAMETERS

The concept of an "object-oriented" operating system has received much attention in the literature in recent years (e.g., see the papers by Jones (1978) and Rattner and Cox (1980)). In SD/SDM, the notion of "object" is used as a unifying term to designate all the software elements in a computer system. Currently, this includes processes, queues, semaphores and data objects. A "process" is an application or operating system program. An instantiation of a process is a "task." Thus, in case a process is reentrant, there can be multiple active tasks corresponding to the process. "Queues" of various disciplines (LIFO, FIFO, etc.) are available for use by the operating system for scheduling tasks, responding to requests or as I/O buffers and message buffers for communication among tasks. "Semaphore" objects are used for task synchronization and "data objects" consist of files, tables, data bases and other types of data structures.

The notion of "parameter" is a key element that allows transfer of information among objects. A parameter is a structured variable which is used to control the behavior of processes. A "parameter set" can optionally be associated with each object and each message generated by a task. The parameter set associated with a process acts as its internal variable set. The parameter set for another type of object or a message acts as a description for it. Thus, a parameter set is a vehicle for separating the size of an object or message and the relevant attributes of it that are needed for performance-oriented decision making. For example, although the structure of a message can be defined in the SDDB component of SD/SDM, for simulation purposes it is considered as a string of bits. To compute throughput or traffic on a bus, the length of the message is all that is necessary. In order not to lose required information, a parameter set can be attached to the message. This parameter set rides freely with the message, i.e., it does not take up any additional capacity on the bus. Since a parameter in the parameter set may be a complex function of the actual message, the ability to represent it independently, greatly simplifies the definition of processes that manipulate messages.

Figure 2 represents a typical flow of control among processes and an "action sequence" within a process. In Figure 2, SA1 is a switch-action or operator transaction that sends a stream of data of size D1 and a parameter set P1 to the process PROG2. Likewise, the process PROG1 sends D2 bits and the parameter set P2 to PROG2. PROG2, in turn, calls PROG3, passing to it D3 bits and the parameter set received from above (indicated by "*"). Within a process, high-level "action-statements" (e.g., READ, WORK, CALL, and variations of the if-statement and while-statement) are available to define what a process does. In addition, since actions are qualified by deterministic and probabilistic conditions, partly defined by the parameters, a great deal of freedom is available in defining the flow of control both within and among processes. For example, most data base management functions and table-driven software can easily be represented through the use of parameters.

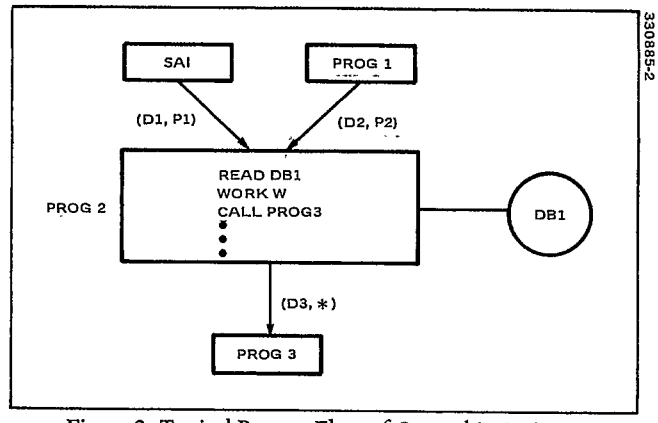

Figure 2: Typical Process Flow of Control in SD/SDM

Beyond the definition of objects and how they interact with each other, one must define where they reside and what the load is. Objects that are allocated to more than one device are defined only once in SD/SDM. This allows convenient analysis of alternate architectures involving the reallocation of objects. The load is defined in terms of an initial start time and an interarrival distribution for transactions like SA1 in Figure 2. Various interarrival distributions can be accommodated in SD/SDM.

4. OPERATING SYSTEM MODEL

One of the basic goals in the design of the current version of SD/SDM has been the capability to model a variety of operating systems. This is accomplished through a kernelized operating system model. Figure 3 shows the relationship of the kernel and the remainder of the operating system model. The kernel, which is the uninterruptible core of the operating system, and the "basic O.S." are hard-coded in the Dynamic Simulation Model (DSM) component of SD/SDM. These functions can be controlled to some extent through parameters which allow, for example, the overhead associated with operating system rou-

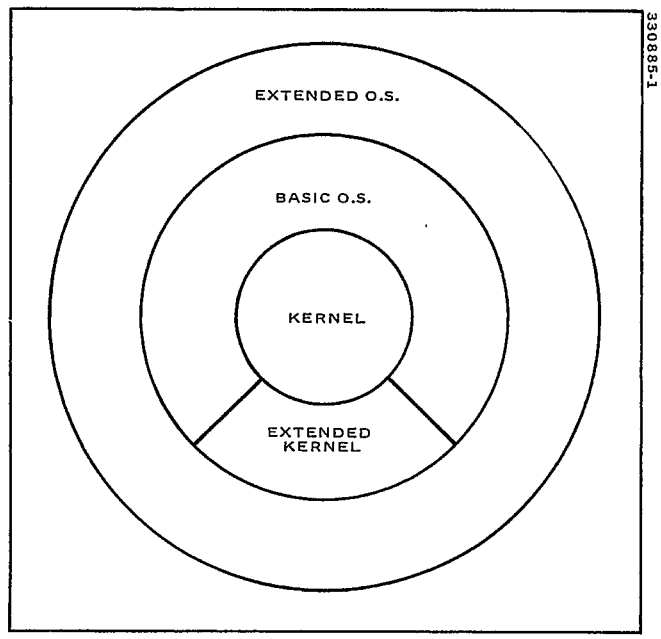

Figure 3: SD/SDM Operating System Model

tines to be defined by the user. A further degree of control is achieved through the specification of an "extended kernel" and "extended O.S." by the user. This provides a wide range of choices in defining task scheduling and dispatching strategies, I/O handling, memory management, interprocess communication and other aspects of an operating system.

The definition of operating system routines are handled in the same manner as application processes. Thus, there is no need for the SIMSCRIPT compiler even for interpreting the actions within operating system processes. There is also the provision to integrate new SIMSCRIPT modules into the model. For these, of course, the compiler has to be used. The only other reason that the compiler might be necessary, is to parse an automatically created program for evaluating mathematical expressions.

The key problems that had to be resolved in the design of the SD/SDM operating system model were the relationship of the kernel to the remainder of the operating system and the exchange of information between SIMSCRIPT and the action language of processes.

An important benefit of the System Design Data Base (SDDB) component of SD/SDM arises in the ability to store the specifications of various operating systems in a design libary. Thus, once an operating system is defined, others may invoke it with a single reference to the appropriate data base. Standard bus protocols such as token passing or carrier sense multiple access with collision detection (CSMA/CD) can also be specified by indicating the appropriate predefined designation for the desired protocol. Thus, the performance of a system using alternate operating systems and bus protocols can easily be studied.

5. APPLICATIONS OF SD/SDM

SD/SDM has been used for the specification and performance evalution of the distributed computer systems associated with a variety of command and control systems. Because of its table-driven input format, very little training has been found to be necessary to begin using it. A highly interactive interface to the SDDB data base is under development now that will make it even easier to use.

As far as the validation of the SD/SDM simulation results is concerned, comparison with actual data in one major project showed maximum error of 6% in processor utilization. More detailed experiments are planned to evaluate the capability of the new operating system model to accurately reflect the behavior of actual systems. Finally, it is expected that once a sufficiently rich design libary has been developed, SD/SDM will become a very powerful tool for the design, as well as, the performance evaluation of distributed computer systems.

REFERENCES

Ambler AL, et al, GYPSY: A language for specification and implementation of verifiable programs, Proceedings of an ACM Conference on Language Design for Reliable Software (1977) 1-10.

Davis AM, The design of a family of application oriented requirements languages, Computer, Vol. 15 (1982), pp. 21 - 28.

Gabrielian A, SD/SDM, a System Design and Static/Dynamic Modeling tool set, Proceedings of the Seventh Annual SAS Users Group International Conference (1982) 44-48.

Jones AK, (1978) The object model: A conceptual tool for structuring software. In Operating Systems, An Advanced Course, R. Bayer, et al (eds.), Springer Verlag, pp. 8 - 16.

Kosy DW, The ECSS II Language For Simulating Computer Systems, Rand Corp. Report R-1895-GSA (1975).

Levitt KN, Robinson L, Silverberg BA, Writing simulatable specifications in SPECIAL, The Use of Formal Specification of Software, H.K. Berg and W.K. Giloi, Editors (1979), pp. 39 - 78.

Rattner J, Cox G (1980), Object-based computer architecture, Computer Architecture News, ACM SIGARCH Newsletter, Vol. 8, No. 4, 4-11.

Standridge CR, Pritsker AAB, An Introduction to the Simulation Data Language, Proceedings of the 1982 Winter Simulation Conference (1982) pp. 617 - 616.