Proceedings of the 1980 Winter Simulation Conference
T.I. Oren, C.M. Shub, P.F. Roth (eds.)

A HIGH LEVEL SIMULATION MODEL OF A NETWORKED COMPUTER SYSTEM

William H. Hochstettler
and
Lawrence L. Rose

BATTELLE
Columbus Laboratories
505 King Avenue
Columbus, Ohio 43201

ABSTRACT: The objective of this research was to design and develop a baseline planning tool for the OCLC networked computer system. This planning tool was necessitated by the dynamic, fast-growing on-line bibliographic needs of OCLC's remote library users. In this paper we show simulation to be the best research approach and justify the IPSS language as appropriate to this application. The resultant model of OCLC's networked computer system is highlighted. This tool was developed in a timely manner to meet the immediate goals of OCLC planners and provides a standard baseline model for future extensions.

1. INTRODUCTION

OCLC, Incorporated is an organization that provides computer-based library services to over 2,200 member libraries in 50 states, the District of Columbia, Puerto Rico and several countries. Computerized services currently offered include a Cataloging Subsystem, Interlibrary Load Subsystem, and Serials Control Subsystem. An on-line Acquisitions Subsystem will be available in late 1980.

1.1 Background

OCLC services are offered through a telecommunications network of dedicated leased and dial-access telephone lines to the OCLC computer center in Columbus, Ohio. Over 1,900 member libraries use special terminals manufactured for OCLC to access the On-line System; other libraries use dial-access terminals. Member libraries can thus catalog books, serials and other library materials, order custom-printed catalog cards, create machine-readable data files, maintain location information on library materials, and facilitate interlibrary lending. Not a single computer, the OCLC On-line System is a unique configuration of heterogenous computers. Due to the expanding nature of the OCLC user community and supporting data bases (the current bibliographic data base has over six million records with an average record length of 560 characters), the computer network has undergone many changes and rapid expansion since its inception in 1971.

Until late in 1978, OCLC used Xerox Sigma computers to do all on-line processing, as shown in Figure 1. As the number of users, terminals, bibliographic services, and data base records increased, OCLC expanded its computer system until it consisted of 12 communications processors feeding the four Xerox Sigma 9 computers that shared access to common memory and the secondary storage media containing the data base. Since further expansion of the data base and processing power by adding more Sigma computers was limited by the multi-access capabilities of the secondary storage media, a major change in the hardware configuration was needed to facilitate expansion.

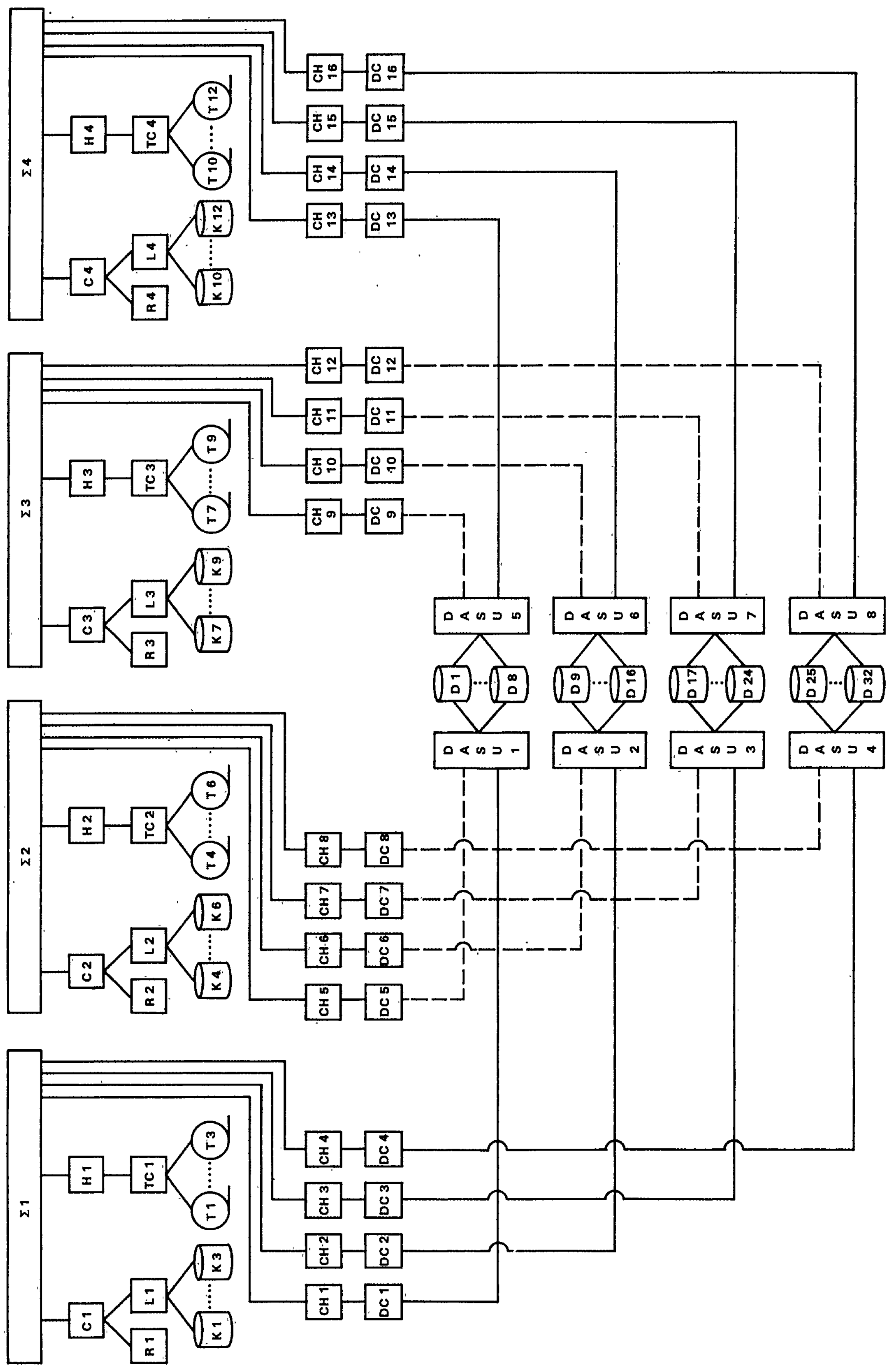

FIGURE 1. OCLC XEROX SIGMA 9 QUAD-PROCESSOR NETWORK

1.2 The Network Configuration

OCLC chose in 1978 to implement a network approach, as shown in Figure 2. The data base function was removed to a separate computer system, the Data Base Processor. Another separate computer system, the Network Supervisor, was added to manage the message transmission between the Communication Processors and the Application Processor (the Sigma 9 computers). Since this network approach was adopted 2 years ago, growth in both secondary storage capacity and computer processing power (so far) has been readily accomplished.

A brief overview of this network is necessary to properly present the problem scenario. The terminals connected to the OCLC On-line System are the primary input source of messages to the system. These terminals transmit data to the OCLC On-line System by an OCLC special ASCII multipoint protocol. The terminals are based on an 8008 microprocessor having 2K bytes of display memory.

The Communications Processor serves as a line concentrator to over 140 2400-baud synchronous multidrop lines. Each processor in the Communication Processor component is a 16-bit word minicomputer with 64K bytes of main storage with 11 2400 bit/sec multidrop lines and dial-access communication lines to the terminals. The Communication Processor is responsible for polling the terminals and transmitting messages between the terminals and the Network Supervisor.

The Network Supervisor and Data Base Processor are both networks of homogeneous minicomputers, the Tandem T16. The Tandem system processors are connected by a high-speed bus which allows them to communicate with each other and to transmit data. The Tandem system offers a mirrored data management facility whereby two copies of a file are maintained at all times. This not only increases reliability but also offers two possible paths to access any record in a file. It is also possible to produce a back-up copy of a file for archival purposes without interfering with any processes currently using the file.

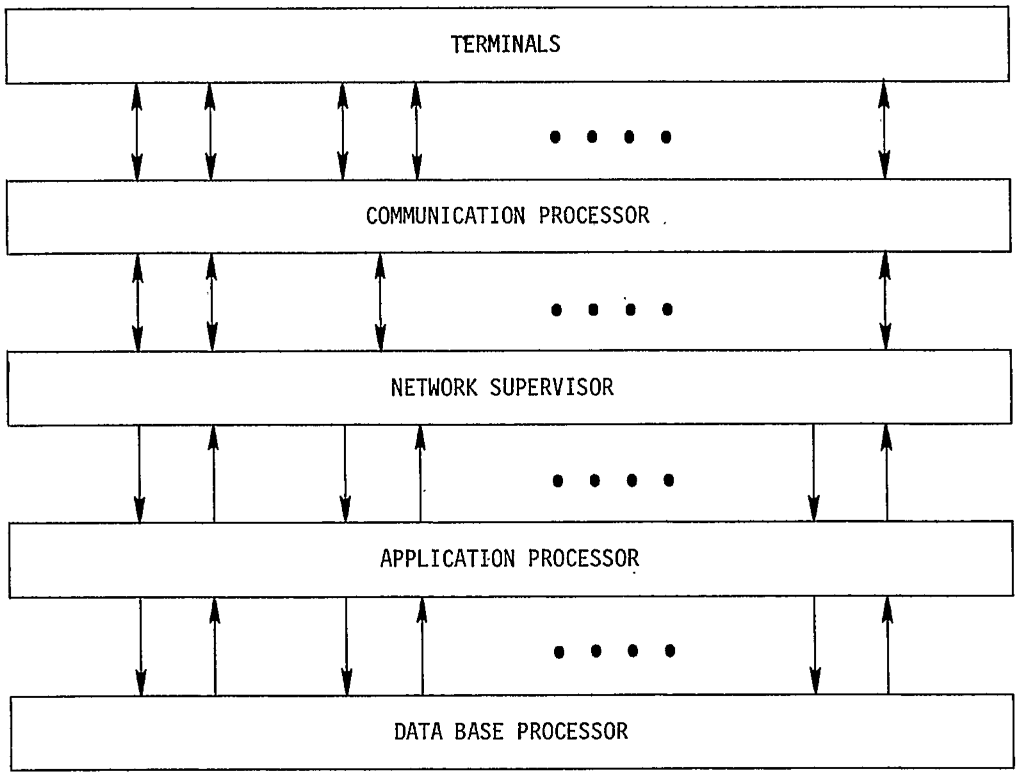

Figure 2. Logical Relationship of the OCLC System

The Network Supervisor currently consists of six Tandem processors, each with from 512K bytes to 704K bytes of main storage. There are 12 synchronous interfaces to the Communication Processor. Five Network Supervisor processors each have two custom interface devices. These usually operate only in one direction, but are capable of bidirectional transmission in case of failure of the other interface in the pair. Each pair of interface devices is connected to an Application Processor.

The Application Processor currently consists of five Xerox Sigma 9 computers. Each central processing unit has 128K words (32 bits) of main storage, some disk secondary storage, three tape drives, a line printer and console. The Xerox computers share two card readers, two card punches, and more secondary disk storage used for off-line data processing. Each Application Processor currently runs three application tasks--cataloging, serials control and interlibrary loan. Each Application Processor is connected through two custom interface devices to the Data Base Processor.

The Data Base Processor is a 14-processor Tandem network. Each processor has at least 864K bytes and up to a maximum of 1.1M bytes of main storage. The entire system has two consoles (one for back), a line printer, and 28 logical 260-megabyte mirrored disk drives for data base storage.

As depicted in Figure 2, the OCLC On-line System functions as follows. Each processor in the Communication Processors has up to 250 terminals connected through leased lines. Eight asynchronous dial-access ports also are available which may vary the load from processor to processor, depending on which terminals are actively being used. A message entered on a terminal has a static path to a Communication Processor, which in turn has a static path to one of six tasks executing on the Network Supervisor to receive inbound messages. The message is transmitted over a static path to a particular Sigma 9 Application Processor. If the message is a request for a data base operation (some messages such as LOG-ON or LOG-OFF do not require Data Base Processor involvement), it is transmitted through a static path to one of six Data Base Communications Processes. The Data Base Communication Process then transmits the request to one of several (currently 13) identical tasks to perform the actual data base access. The response to the inbound message is transmitted to the terminal through the same path just described.

Despite multiple paths possible through the entire On-line System, messages are routed by static paths defined by the OCLC system operators. The only exception is the input/output tasks of the Data Base Processor, which run on several central processor units. Thus, the tasks in the various components of the On-line System are statically connected for message transmission and are modeled in a similar manner.

2. PROBLEM STATEMENT

The OCLC support system, both hardware, software, and data base, has grown dramatically in its first decade. With the recent trend towards on-line cataloging and interlibrary loan and the constant growth of published materials, the OCLC workload continues to increase, with no plateau in sight. The hardware and software support systems are becoming more and more complex and sophisticated by necessity. Planning for expansion to incorporate more users, increased traffic from each user, and more library services is a must. Yet the impact of migrating to new hardware configurations or utilizing different data base organizations or reassigning software processes to the cpu's in the OCLC network becomes increasingly more difficult to predict or assess.

Within this dynamic environment OCLC must remain competetive with other on-line library facilities. Competetive factors include: (1) Cost per user transaction (for instance, the cost of a catalog card or authority reference)--relates to overall hardware cost and cpu time; (2) Data base coverage (one cannot charge for a "not found" answer)--relates to storage size and organization; (3) Variety of services available to the client--relates to robustness of software processes; and (4) Response time--performance characteristics of the system must be maintained. Response time can be singled out as critical: once a user is accustomed to 5 seconds response, a 20 second response becomes unacceptable. Thus, OCLC's growth in data base size, services, and customers must not sacrifice response time.

A clear need exists for a planning tool for management decision making at OCLC: a tool which, in a cost effective and timely manner, can be utilized to assess the impact of suggested system changes to the behavior of the OCLC On-line System at a global level.

3. SIMULATION AS A PLANNING TOOL

To date, the major tool for evaluating system changes at OCLC has been experimentation with the actual system, either off-line after normal production hours or on-line during production hours. Experimentation lowers the reliability of the system and requires time and effort to implement proposed changes. If a change is determined unsatisfactory, the effort is a loss. Therefore, a more reliable tool for planning and evaluation at all stages of system use was required.

Analytic models are often utilized to characterize queueing systems (Gross and Harris, 1974), but with varying degrees of success. While an analytic model (if derivable for OCLC's system) would be effective in assessing growth and change in user traffic and usage rates, it could not readily cope with modifications to consider data base organization changes, additional software processes to support new on-line uses, hardware reconfigurations or upgrades, etc. The fact that the OCLC system is unique and has not reached steady state further complicates matters (Graybeal and Pooch, 1980).

Simulation models and languages, both continuous and discrete, thus received prime consideration by the authors. Planning is one of the primary objectives of simulation, as noted by author R.E. Shannon (1975):

> "Simulation is the process of designing a model for a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criteria) for the operation of the system."

As an alternative to actual system experimentation, simulation offers several distinct advantages:

(1) Cost - Construction and use of a simulation model is considerably less costly than actual experimentation, in terms of acquisition costs, operational costs, and manhours.

(2) Flexibility - A properly designed and parameterized simulation model enables the decision-maker to explore several alternative solutions; actual experimentation offers limited flexibility.

(3) Timeliness - A simulation model enables timely consideration of alternatives, and timely data is information of use to the decision-maker.

(4) Ease of Use - A good simulator provides an unambiguous definition of the process being modeled. The more closely the user understands the basic concept of the model, the more easily he can use it and have confidence in its results.

Over the past 15 years, simulation has become better understood and utilized as a planning tool. Successful-related implementations, not only at OCLC (Wong, 1979), but at large agencies such as the U.S. Army (Brownsmith, Carson, and Hochstettler, 1979, Rose, 1979) and the U.S. Social Security Agency (Maisel and Gnognoli, 1972, Rose and Carr, 1979) have paved the way and provided a firm basis for further implementation of simulation models.

4. CHOOSING AN APPROPRIATE LANGUAGE

Computer languages provide capable and flexible support for simulation activities that are characterized by properties such as complexity, magnitude, time variance, and repetition. These languages are generally classified into three categories: general purpose, continuous simulation, and discrete simulation.

General purpose languages, such as FORTRAN, PL/1, and ALGOL have been extensively used in simulation. However, they offer no particular environment or "world views" for the modeler--one must build the model from scratch. As simulation languages have increased in both number, reliability, and sophistication, they have become more attractive to the modeler than the general purpose languages. For this research activity, timeliness and flexibility were requisites in the modeling process; hence an approach that would reduce the programming requirements was preferred.

Continuous simulation languages, such as CSL, DYNAMO, and CSMP offer the modeler an environment particular to the solution of user-defined difference or differential equations. The specification of a computer network, its users, its software, and its workload as a set of difference or differential equations is exceedingly difficult and can require simplifying assumptions not desired in a high level of detail. These languages are not particularly suitable for computer systems simulation.

4.1 Discrete Simulation Languages

Discrete simulation languages have been under development for over 15 years, and in the last 5 years they have proved extremely valuable to the modeler. They provide constructs such as a time clock, next events queue, data structures, statistics, dynamic transactions, etc., automatically to the modeler. Thus, one can, when utilizing one of these languages, concentrate on the definition of the model as opposed to the implementation of the model on a computer in some alien language. One defines a discrete simulation model by detailing all of the events/acts pertinent to the system and the actions which trigger them. The model is defined at a macro or micro level depending on the detail of the design or the requisite precision of the output. Four types of discrete simulation languages can be distinguished: (1) Activity-oriented, (2) Event-oriented, (3) Process-oriented, and (4) Transaction-oriented.

Activity-Oriented Languages

Activity-oriented languages represent time-dependent activities as instantaneous occurrences in simulated time. Thus, one does not schedule occurrences, rather one specifies under what conditions they can happen. The modeler program is composed of two major sections: a test section to determine what activities can now occur, and an action section to update state and time conditions. The language implicitly controls the invocation of the test and action sections. CSL (Buxton, 1966) is an example of an activity-oriented language.

Event-Oriented Languages

Event-oriented languages represent an event as an instantaneous occurrence in simulated time, automatically scheduled to occur when it is known by the model definition that the proper conditions exist for its occurrence. All events and their interactions are defined independently; an executive program can automatically sequence all scheduled events for the modeler. GASP (Pritsker, 1975), a FORTRAN extension, and SIMSCRIPT (Kiviat, Villanueva, and Markowitz, 1975) are the two main event-oriented simulation languages used today.

Process-Oriented Languages

Process-oriented languages are a hybrid derived from the concise notation of activity-oriented languages and the efficiencies of event-oriented languages. A process is a set of events; it is dynamic and can exist over time. Processes can be interrupted, have subprocesses, and can be reactivated. The executive program controlling these processes is necessarily more complex than that required for event/activity-oriented languages. SIMULA (Dahl, Nygard, 1966) is an ALGOL extension that exemplifies the process orientation.

Transaction-Oriented Languages

Transaction flow-oriented languages use the block structure of flow charts to describe the simulation, with transactions flowing through activity and time altering blocks. Each block specifies specific actions, with restrictions on parallel execution, etc. These languages are extremely easy to use but much less flexible than the other types. GPSS (Gordon, 1975) is the best known language of this type, but recently SLAM (Pritsker and Pegden, 1979) was introduced as a transaction-oriented extension to GASP.

4.2 Desired Language Features

Given this wide range of discrete-event simulation languages, how does one determine that subset of languages that is suitable for ones needs? Mihram (1972) notes five desirable features for the chosen subset as follows:

(1) Ability to characterize initial model status
(2) Flexibility in model definition
(3) Report and statistical generation capability
(4) Design flexibility for validation and analysis, and
(5) World view consistent with modeling views of process.

A sixth notable feature is added by Pritsker (1979):

(6) Self-documentability of the code.

From our standpoint of computer systems simulators, the "world view" is paramount; yet none of these features can be ignored.

A quick examination of the four language types revealed that, for this problem domain, the process or event-oriented languages were preferable to the transaction or activity-oriented languages such as GPSS and CSL. The process orientation of SIMULA is very attractive due to our view of a computer system as one of interacting, interruptible processes--including manual, software, and hardware. System priorities create hierarchies of control, and those, too, are clearly modeled by processes and subprocesses. The event orientation of SIMSCRIPT or GASP provides code modularity and varying levels of detail. Unfortunately, the events form a flat rather than a hierarchial control structure as required. These three languages satisfy Mihram's aforementioned initial four criteria reasonably well, but the world view of each language is extremely general. And the self-documentability of the code is questionable. These deficiencies are due to the fact that none of these languages is computer-systems oriented. There are no language statements that directly relate to hardware, software, data, jobs, etc.; thus, the modeler faces a stern test to construct a valid model of a computer system given only these general simulation tools.

4.3 Special Purpose Simulation Languages

Recent research has led to the development of several simulation languages specific to computer systems simulation. Three of these languages are: SIMTRAN (Wong, 1975), ECSS (Kosy, 1975), and IPSS (DeLutis, 1978). It is notable that SIMTRAN and ECSS are extensions to event-oriented GASP and SIMSCRIPT, respectively, and IPSS is an ALGOL-like process-oriented language. Thus, these three languages are in our preferred class of process/event-oriented tools. Each provides initialization characterization, flexibility in definition, report generation, and design flexibility albeit in differing forms.

Besides satisfying the criteria of their predecessors, these languages provide a world view in light of computer systems and code that relates to computer science terminology. Thus, we find these specialized languages to satisfy the last two criteria (which the general purpose simulation languages failed to do). These languages enable one to describe a computer-based information system in a meaningful and straightforward manner with considerably less effort than would otherwise be possible.

SIMTRAN is notable for its hardware characterization, ECC for its software and operating system characterization, and IPSS for its data base characterization. Each has components to model all aspects of an information system, but their capabilities and inherent levels of detail differ significantly.

The IPSS language was chosen for this project for several compelling reasons. First, SIMTRAN is a proprietary product of General Electric and available only on their user network. Second, ECSS requires the SIMSCRIPT compiler which was unavailable on the OCLC Sigma system configuration. Lastly, OCLC has had some prior successful experience using IPSS to model OCLC applications and encouraged its further use.

5. THE IPSS LANGUAGE

The Information Processing System Simulator (IPSS) is a discrete event simulation language with special features to support the modeling of information systems. While the IPSS is capable of concisely modeling complex information systems, it is general enough in nature to model naturally occurring systems. Special attention to the hardware components and the data management facilities of a computer system and data base structures makes IPSS especially appropriate for information system simulation.

5.1 The Methodological View

The specification of a model in IPSS is a three-step-transformation process as depicted in Figure 3. The first step is the transformation of knowledge about the system into four logical components (services, processors, information stores, and user workload). The components may be specified precisely or imprecisely. A combination of levels of detail enables the focus of the model to be on the modeler's areas of special interest. This component specification does not use a simulation language, but is a methodological division of the system into a logical component structure for modeling. It provides a meaningful interface between the user and the modeler.

Services Component
The specification of the system tasks is accomplished in the services component. These services may be manual tasks or software modules of a computer information system. These are the processes that are invoked by the job stream (workload component).

Hardware Resource Component
The hardware resource component identifies those computer, human, and other processors available to do the system's work. It describes the physical means for information transmission and storage.

Information Storage Component
The information storage component is the specification of the logical relationships between data and the physical storage system. This logical-physical linking is one of IPSS' strengths.

5.2 IPSS Model Formation

The second transformation process takes the four logical component descriptions and produces four components written in the IPSS language. This process requires a simulation modeler with IPSS technical expertise. The model specified in the IPSS language consists of up to four separate components: (1) Configuration, (2) Exogenous Event, (3) Data Base, and (4) Model.

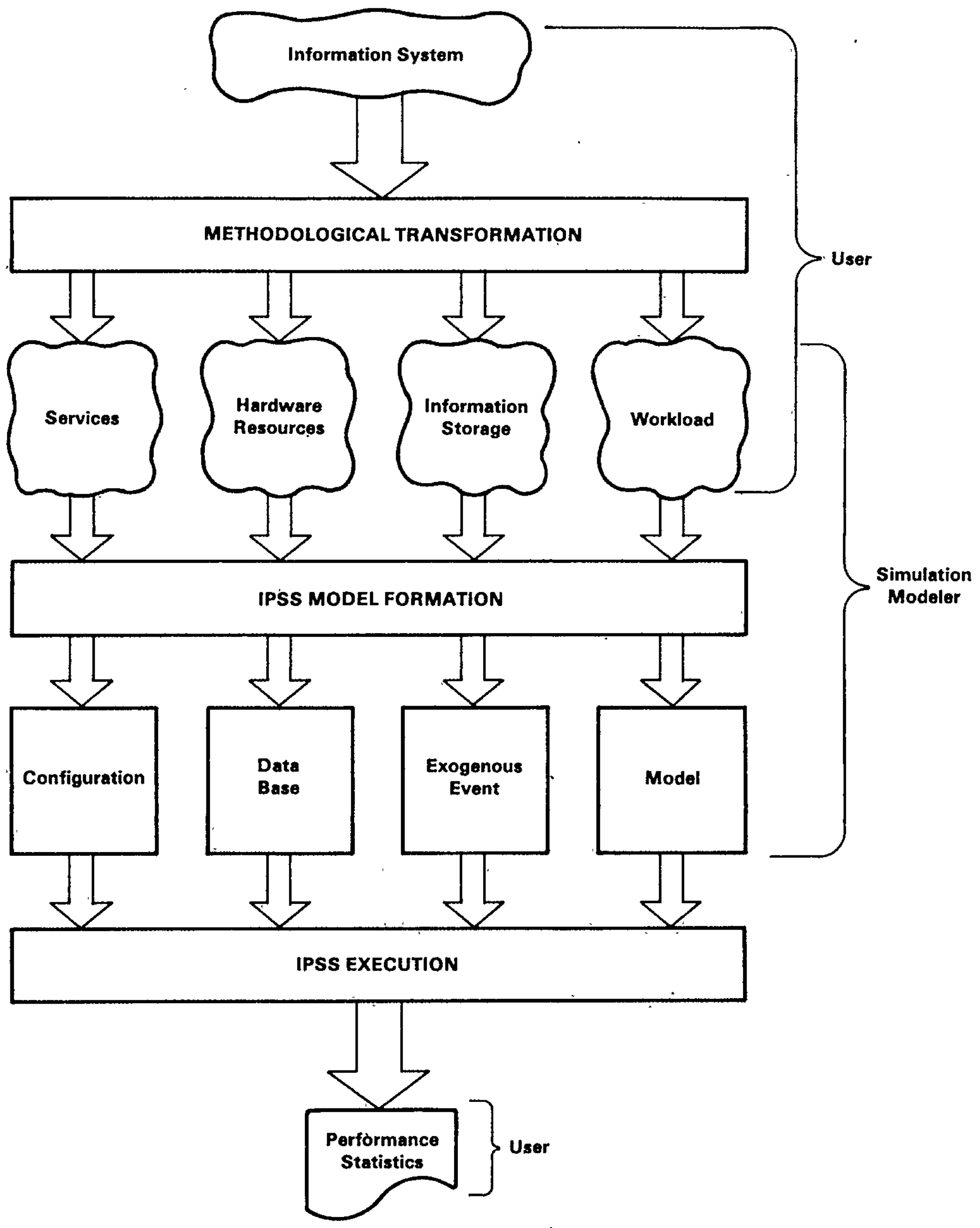

FIGURE 3. MODELING METHODOLOGY

Configuration Component
The Configuration Component defines the hardware elements and software services of the information system. The IPSS has several special facilities that allow the modeler to gather statistics on the usage of the facilities. Particular hardware devices may be named, such as central processor, data channel, and secondary storage devices (Appendix A). Special queues or control points also may be defined.

The Configuration Component defines the exogenous and endogenous services that constitute the model. An exogenous service is invoked by the occurrence of an external event (such as the arrival of a message), while an endogenous event is invoked as a consequence of the exogenous event (such as the processing of the message just arrived). These two types of services, as defined in the IPSS language, may contain special IPSS statements that allow the queueing, seizing, and releasing of named facilities, as well as the creation and destruction of files. Fortran code intermixed with the IPSS statements provides further processing, control, or input/output.

Exogenous Event Component
The Exogenous Event Component defines the information system's service requests. Here the modeler describes the distribution of the occurrences of the events that will drive the model. This service component is later mapped to the exogenous services defined in the Configuration Component.

Data Base Component
The Data Base Component defines the various files of the information system (Appendix A). Extensions to the IPSS to allow the definition of data base schemas has been proposed but not implemented (Brownsmith, 1979). These definitions are later mapped onto the physical devices defined in the Configuration Component.

Model Component
The final section of an IPSS model, the Model Component, allows the modeler to associate exogenous events with exogenous services. Files from the Data Base Component are assigned devices within the Configuration Component. The ability to equate these entities from the different components allows the modeler to substitute different components as the model changes and experimentation proceeds. Finally, the modeler specifies the options for the model's execution, including halting criteria, trace options, and statistical generation and reporting.

5.3 IPSS Execution

The final transformation depicted in Figure 3 is the execution of the model by the IPSS Execution Facility to produce performance statistics. The statistics can then be evaluated by the system modeler for validation and by the simulation modeler for verification.

5.4 IPSS Model Parameters and Output

The user of the IPSS model provides a workload for the system specified by the interarrival time of exogenous events. Changes to the hardware components or services may be made to investigate a proposed solution to a problem. The IPSS simulation facility produces up to eight classes of statistics at the conclusion of the simulation; some statistics are provided automatically while others are user-triggered by statistical collection statements. The classes of statistics produced are:

(1) General statistics
(2) Request stream statistics
(3) I/O activity statistics
(4) Queueing statistics
(5) Utilization statistics
(6) Wait statistics
(7) Service statistics, and
(8) Task/activity statistics.

6. THE BASELINE MODEL

Despite the existence of detailed models of the OCLC System (Wong, 1979, and DeLutis, Wong, Rush 1979), a new approach was needed to produce a model general enough to be changed easily by the system modeler, and yet still capable of being verified and validated. Previous models had included the storage structure of the OCLC data base in great detail. However, the new OCLC model represents the major On-line System components with the capability of adding further detail uniformly or to a particular component of the system, depending on the interest and modeling needs of the modeler. This high-level model was designed for construction and validation, particularly since statistics on the On-line System's performance are already compiled and available at a level of detail commensurate with the model.

Consistent with the methodology of the previous section, the major systems of the OCLC On-line system were identified for modeling. The logical components were initially identified by hardware function. Referring to Figure 2, five classes of hardware are illustrated (Terminals, Communication Processors, Network Supervisors, Application Processors, and the Data Base Processors). An initial model consisted of a service for each of the subsystems.

Several of the logical components have major software programs that have responsibility for a single task. For example, the Network Supervisor consists of a DYNCIO task that synchronizes message transmission to and from the Communication Processors. The CNIO task handles message transmission to and from the Application processors. Thus, the Network Supervisor is modeled by two services.

The OCLC On-line System is message driven from the terminals. The workload is defined by the number and frequency of messages generated. As the arrows indicate in Figure 2, messages progress through the system components from the terminal to the data base processor and back to the terminal.

Now that the methodological transformation is complete, the model can be translated into IPSS. The model is constructed by top-down refinement. First, a service that describes the system operation in terms of logical pathways through the components is defined. Then, each system component is defined by one or more additional services. Later, each component service may be expanded or split into more subcomponent services adding detail at each level. Using this technique with the IPSS, one can describe the system by a straightforward mapping from the actual system modules to services in the model.

The OCLC model has one major exogenous service which traces the transmission of the message through the system by queueing, seizing, invoking, and releasing the various software tasks that are defined as endogenous services in the model. Thus, each OCLC On-line System component--Communication Processor, Network Supervisor, Application Processor, and Data Base Processor is modeled as one or more endogenous services.

6.1 Hardware Definition

Models in the IPSS are organized into components describing the environment or facilities of the information system and services provided. Defining the computer hardware is straightforward. Shown below is a portion of the Configuration Component which defines the central processing units of the OCLC network: CP, NS, Host, and DBP are the names of the central processing units for the Communication Processor, Network Supervisor, Application Processor, and Data Base Processor, respectively.

```
CENTRAL PROCESSOR:  ID=(CP,15);
CENTRAL PROCESSOR:  ID=(NS,6);
CENTRAL PROCESSOR:  ID=(HOST,5);
CENTRAL PROCESSOR:  ID=(DBP,14).
```

Note that more than one central processing unit is defined in each case. As the system expands by adding more central processing units, the number in the definition is increased. All facilities can be defined in the the same manner.

6.2 Information Services Definition

The definition of the services in the IPSS follows the natural organization of the information system being modeled. As previously stated, the services are divided into two classes--exogenous and endogenous. The exogenous services handle the events that trigger the system, and the endogenous services describe these operations that result from the system being activated.

Figure 4 illustrates the Data Link Control (DLC) task which is the software module in the Communication Processor that transmits messages between the terminals and the Network Supervisor.

```
ENDO SERVICE:  ID=DLC, NAME=DLC$, PARAMETER LIST=(DLCNOX),
     SAVE AREA SIZE=9;
     INTEGER DLCNO, DLCNOX
     EQUIVALENCE ($SAVE(4),DLCNO)
     INTEGER AVAIL/0/, INUSE/1/
END: DECLARATIONS:
C    SAVE THE DLC NUMBER FOR THIS INVOCATION
     DLCNO=DLCNOX
END: INITIALIZATION;
C    COLLECT UTILIZATION STATS ON THIS ENDO SERVICE.
     SEIZE:          FACILITY=DLCTCP(DLCNO);
C    QUEUE TO GET CP CPU BEFORE TRANSMITTING TO NS.
     QUEUE:          FACILITY=CP(DLCNO);
     WAIT FACILITY:  FACILITY=CP(DLCNO),      STATUS=AVAIL;
     DEPART QUEUE:   FACILITY=CP(DLCNO);
     SET STATUS:     FACILITY=CP(DLCNO),      STATUS=INUSE;
     SEIZE:          FACILITY=CP(DLCNO);
C    TRANSMIT MESSAGE TO NS.  (ASSUME .02 MILLISECOND)
     PROCESS:                TIME=.2E+1, CENTRAL PROCESSOR=CP(DLCNO);
C    FREE DLC CPU AND RETURN.
     RELEASE:        FACILITY=CP(DLCNO);
     SET STATUS:     FACILITY=CP(DLCNO),      STATUS=AVAIL;
     RELEASE:        FACILITY=DLCTCP(DLCNO);
END: ENDO SERVICE:
```

FIGURE 4. DLC ENDOGENOUS SERVICE

The endogenous service DLC is seized and released so that the IPSS will provide utilization statistics at the end of the simulation. The processor number of the Communication Processor that this DLC task runs on has been determined previously by the invoking service and is passed as a parameter. The DLC task advances the clock a fixed time quantum to simulate the processing time expended on the message transmitted. However, a distribution of times could be sampled or statements could be substituted which actually model the message processing the DLC performs. Another possibility would be to replace the PROCESS statement with an invocation of a lower-level endogenous service. Thus, a natural model of the actual system operation has been synthesized by the division of the DLC task into an input/ output task and message processing task.

6.3 Workload Definition

The exogenous event stream which defines the arrival of the events that trigger the exogenous services defined in the configuration section consists of one major event--the generation of a message at a terminal. In this model the arrival time is based on peak time usage when a message enters the system (on the average every .03 seconds). Figure 5 is the IPSS listing of this exogenous event procedure, which is translated into a Fortran subroutine, as are all IPSS procedures.

```
EXOGENOUS EVENT:     ID=EXMSG,                 TIME=PROC(EXMSGS);
PROCEDURE:                     NAME=EXMSGS,         TYPE=SUBROUTINE;
     COMMON /MSGCNT/ MSGCNT, MIGTRC
     INTEGER MSGCNT
     LOGICAL MSGTRC
     REAL*8 ITIME
C    START AT 9.00AM  (9*60*60*10000)  --  TIME IS IN TENTHS OF MS.
     DATA ITIME/32400.0D+4/
     SYSCOM(1) = 0
C *** A MESSAGE ARRIVES ON THE AVERAGE EVERY .03 SECONDS.
C    STORE INTERARRIVAL TIME OF MESSAGES IN TENTHS OF MILLISECONDS
     ITIME = 0.3D+4

     CALL $SREAL(ITIME,SYSCOM,34)
     RETURN
END: PROCEDURE;
END: EXOGENOUS EVENT STREAM:
```

FIGURE 5. IPSS EXOGENOUS EVENT ROUTINE

Extension is straightforward. The constant average time could be replaced by a sampling of a distribution, or the clock could be examined and a different distribution used for nonpeak system loading. Further, another exogenous event could be supplied that would change the distribution table this exogenous service uses. Thus, a separation between the occurrence of the event (exogenous event routine) and the action from the occurrence (processing of the exogenous service) is established in the IPSS language. This separation promotes natural modeling and allows the modeler to experiment by simply changing a small exogenous event stream procedure independent of the actual processes the exogenous event triggers.

7. MODEL UTILITY

Since the OCLC On-line System is a dedicated real-time computer system, the primary performance measure is response time. OCLC corporate policy mandates a minimum response time. This high-level model produces this vital statistic in addition to the intermediate statistics such as queueing times and facility utilizations. Despite the lack of great detail, the model is valuable for experimenting with system changes that can be succinctly expressed. The most obvious example is a change in computer hardware, for which the model would immediately provide statistics for evaluation. These statistics are provided without endangering the safe operation of the On-line System and can be done before purchasing new equipment, thereby allowing cost/benefit analysis based on performance modeling.

Changes in the OCLC On-line System are generally the result of changes in the workload, types of services, or types and number of processors. This baseline model can be readily modified to support these types of changes. The top-down design of the model necessitates changes in only these services affected without disturbing the rest of the model.

The user workload is defined in the model by the service representing the terminals. The baseline model contained a constant interarrival time; however, another service can be written to model the user actually keying the message. The service can also be altered to reflect changes in system usage. Users may enter fewer extended searches because the system penalizes such requests with long response times.

The OCLC services change as more features to the On-line System are added. In 1979, the Interlibrary loan subsystem became operational. The addition of this new service is reflected in changes to the service modeling. The application processors, and an addition to the Data Base Processor Service to reflect the Interlibrary loan data base.

Changes in processors are typical of system upgrades. Periodically, more processors and data base storage units are added as the system usage and data base grows. This is done by merely increasing the number of units in the hardware description as described in Section 6. This is done without disturbing the definition of the services which use these hardware components.

A more complex change may involve the addition of both new hardware and software. A new subsystem may be implemented on a different brand of host computers. (Currently, all subsystems are executing on all host computers.) The new computer is added to the hardware description and a service is rewritten to simulate the new subsystem. Minor modifications are made to the services simulating the Network Supervisor tasks to recognize the new message types requesting the new subsystem processing and to route them to the new host computer definition for simulated execution.

It is also possible to extend the model of the application tasks as new application subsystems are in development and planning. The IPSS model would produce a system impact statement on these new applications before their installation.

8. CONCLUSIONS

This research has demonstrated the feasibility of implementing a model of a complex information system running on an advanced networked computer system such as OCLC's, in the IPSS language. Due to the special features of the IPSS, the model can be developed faster and with less modeler effort than by using a general purpose simulation language. The model written in IPSS is self-documenting and provides the modeler with greater understanding of the system being modeled. This research further points to the possibility of expansion of the model to support not only system performance and evaluation due to the growth of services (new library applications), changes in user habits, and growth in user population, but also modeling the user driving this on-line system in more detail than just the arrival of a message every so many seconds.

Several features of the IPSS were found most useful in constructing the OCLC model. An IPSS endogenous service can be treated as a facility and thus, it can be queued, seized, invoked and released. This was particularly helpful in modeling the many identical tasks running on the multiple central processing units of the On-line System. For example, the computers in the Applications Processor each have a copy of the same tasks to do the application processing. The IPSS further allows the definition of Task Control Points. These are used as the facility name that is queued, seized and released, thereby allowing one definition of a service to be modeled as being available as copies on many central processing units.

The goal of this OCLC On-line System model was to provide a prototype model for further development based on the analytical needs of system planners. The model's scope included all major system components (both hardware and software) at a "black box" level of detail to provide a broad and general basis for future modeling endeavors.

Due to the level of the model and the general availability of the information on the system, the model was quickly developed and verified within a 3-month period. The design and implementation of this compact model was enhanced significantly due to the appropriateness of IPSS to the modeling of information systems.

The production of the model demonstrates the validity of high-level simulation of a network computer system to system analysts charged with the creation and maintenance of the On-line System. The model is appropriate for tuning, planning, system analysis, and design. Further, the modeling facility is capable of reacting swiftly to changes both of an actual or experimental nature.

Although the current version of this OCLC model does not make use of the data base features of the IPSS, it will be possible at a later time to extend effectively the Data Base Processor endogenous services defined in the IPSS language to model the OCLC data bases. Further extensions may include the actual modeling of the various operating systems running on the separate components--Network Supervisor, Application Processor, and Data Base Processor. This promises to be a powerful planning tool and should aid the OCLC system engineers in maintaining a viable system to support OCLC's needs, both today and in the future.

REFERENCES

Brownsmith, J.D., J.S. Carson, and W.H. Hochstettler (1979), An IPSS-Based Model-Building Methodology for Ranking and Evaluating Computer Hardware/Software Systems, Final Report to AIRMICS, Georgia Institute of Technology, 162 p.

Brownsmith, J.D. (1979), A Methodology for the Performance Evaluation of Data Base Systems: An Extension of the IPSS Methodology, Ph.D. Dissertation, The Ohio State University, 242 p.

Buxton, J.N. (1966), "Writing Simulations in CSL", Computer Journal, IX, 2, pp. 137-143.

Dahl, O. and K. Nygaurd (1966), "SIMULA: An ALGOL-Based Simulation Language", CACM 9, 9, pp. 671-678.

DeLutis, T.G. (1978), The Information Processing System Simulator (IPSS): Language Syntax and Semantics, OSU-CISRC-TR-78-6, The Ohio State University, 477 p.

DeLutis, T.G., J.E. Rush, and P.M.K. Wong (1979), "The Modeling of a Large On-Line, Real-Time Information System", Proceedings of the 12th Annual Simulation Symposium, pp. 350-370.

Ferrari, D. (1978), Computer Systems Performance Evaluation, Prentice-Hall, Englewood Cliffs, New Jersey, 554 p.

Gordon, G. (1975), The Application of GPSS V to Discrete Systems Simulation, Prentice-Hall, Englewood Cliffs, New Jersey, 389 p.

Graybeal, W.J. and U.W. Pooch (1980), Simulation Principles and Methods, Winthrop Computer Systems Series, Cambridge, Massachusetts, 249 p.

Gross, D. and C.M. Harris (1974), Fundamentals of Queueing Theory, New York, New York, John Wiley & Sons, 556 p.

Kiviat, P.J., R. Villanueva, and H.M. Markowitz (1975), SIMSCRIPTII.5 Programming Language, E.C. Russell (ed.), Los Angeles, California, 384 p.

Kosy, D.W. (1975), The ECSS II Language for Simulating Computer Systems, R-1895-GSA, RAND, Santa Monica, California, 472 p.

Maisel, H. and G. Gnugnoli (1972), Simulation of Discrete Stochastic Systems, Science Research Associates, Chicago, Illinois, 465 p.

Mihram, G.A. (1972), Simulation: Statistical Foundations and Methodology, Academic Press, New York, New York, 526 p.

Pritsker, A.A.B. (1975), The GASP IV Simulation Language, Wiley-Interscience, New York, New York, 451 p.

Pritsker, A.A.B. and C.D. Pegden (1979), Introduction to Simulation and SLAM, Halsted Press, New York, New York, 588 p.

Rose, L.L. (1978), Computer Systems/Database Simulation, Final Report to AIRMICS, Georgia Institute of Technology, 50 p.

Rose, L.L. and G.W. Carr (1979), Modeling the SSA Process, OSU-CISRC-TR-78-7, The Ohio State University, 51 p.

Rose, L.L. (1979), "IPSS: A Language and Methodology for Information Processing Systems Simulation", Proceedings of the Simulation-Management Workshop, AIRMICS, pp. 142-174.

Shannon, R.E. (1975), Systems Simulation: The Art and Science, Prentice-Hall, Englewood Cliffs, New Jersey, 387 p.

Wong, G. (1975), "Computer System Simulation with GASP IV", Proceedings of the 1975 Winter Simulation Conference, pp. 205-209.

Wong, P.M.K. (1975), A Methodology for the Definition of Data Base Workloads: An Extension to the IPSS Methodology, Ph.D. Dissertation, The Ohio State University, 269 p.

APPENDIX

IPSS Definitional Statements

- Configuration Component

  Access Mechanism
  Buffer Pool
  Central Processor
  Control Unit
  Data Channel
  Data Set
  Device
  Endogenous Service
  Exogenous Service
  Procedure
  Queue
  Semaphore
  Volume

- Exogenous Event Component

  Exogenous Event
  Procedure

- Data Base Component

  Area
    Segement
  Organization Method
    Extent
    Record Type
  Device
  Procedure
  Volume

- Model Definition Component

  Begin .. End Model

IPSS Procedural Statements

- Configuration Component

  Allocate Data Set Extent
  Create Data Set
  Destroy Transactions
  Find in Queue
  Get Record Address
  Post Semaphore
  Process Time
  Read Physical Record
  Request Service
  Rewind Tape
  Seek DASD Cylinder
  Set DASD Track Sector
  Set Facility Status
  Set System Parameter
  Start Queue Statistics
  Start Usage Statistics
  Stop Queue Statistics
  Stop Simulation
  Stop Usage Statistics

- Model Definition Component

  Equate Facility
  Simulate