Proceedings of the 1985 Winter Simulation Conference
D. Gantz, G. Blais, S. Solomon (eds.)

COMPUTER PERFORMANCE EVALUATION WITH GIST: A TOOL FOR SPECIFYING EXTENDED QUEUEING NETWORK MODELS

J.B. Sinclair
K.A. Doshi
S. Madala

Department of Electrical and Computer Engineering
Rice University
Houston, TX 77521-1892

ABSTRACT

GIST (Graphical Input Simulation Tool) is a powerful and user-friendly performance evaluation tool for the specification and simulation of Extended Queueing Network models of computer and other systems. It includes a number of modeling abstractions for active and passive resource management, job creation and destruction, synchronization, routing, and statistics collection. Two interfaces are provided: a graphical interface (GUIDE) that allows the user to describe much of the simulation model pictorially; and a textual interface (TIDE) that has the same modeling capabilities but uses a menu-driven, window-oriented approach for use on non-graphics terminals. Object and run-time parameters in both interfaces are specified through menus that reduce the probability of specification-time errors. A number of extensions planned for GIST are also described.

1. Introduction

The Graphical Input Simulation Tool (GIST) developed at Rice over the last two years is a performance evaluation tool that allows computer (and other) systems to be modeled as extended queueing networks (EQNs) and evaluated through discrete event simulation. The tool has several unique features, both in its modeling capabilities and in the user interfaces, that distinguish it from other efforts with similar objectives. The software package that comprises the tool is continually evolving, and we discuss its present status and future directions.

In the process of evaluating a computer system via simulation, the system analyst constructs an abstract model of the system. This model is then translated into an appropriate computer program for execution, often by someone who is more familiar with the mechanics of specifying the model but who has little or no knowledge of the system being modeled. The separation of tasks introduces delays and potential errors. Ideally, the analyst should be able to work with a "smart" tool that is easy to use, has powerful features to permit accurate modeling of relevant system characteristics at appropriate levels of abstraction, and can automatically generate the executable simulation code from the high-level model specification.

In GIST, EQN models are specified as a network of objects drawn from a set of predefined object types. The major parts of GIST are: (1) two user interfaces -- a graphics-oriented interface and a textual, dialogue-oriented interface -- that accept specifications from the user, (2) a translator that generates source code for a simulation compiler, and (3) a library of object routines modeling the various object types (Fig. 1). GIST is based on CSIM, a package that provides runtime support environment for discrete event simulation[1]. At present, the graphics interface is implemented on the Macintosh personal computer. The textual input interface allows model specification through an interactive dialogue editor on non-graphics terminals.

GIST is an example of a transactions-oriented modeling tool, in that the model is specified as a collection of instances of predefined object types which interact with one another through jobs which are routed from one object instance to another. Other examples of transactions-oriented modeling and analysis or simulation tools for computer systems are RESQUE[2], QNA[3, 4], STEP-1[5], QNAP2[6], COPE[7], XL[8], SNAP[9], SuperNet[10], PANACEA[11], PERFORMS[12], BEST/1[13], NUMAS[14], the Performance Analyst's Workbench System (PAWS)[15], PLANS[16], and the Performance Analysis Workstation[17]. Graphics interfaces for model specification and/or result display have been considered for or implemented in several systems, including RESQUE[18], PAWS[19], NUMAS[14], and the Performance Analysis Workstation[17]. A brief survey of most of these tools can be found in [20].

2. User Interfaces

Users can specify an EQN model for a computer system through either a graphical interface or a textual interface. Conceptually both interfaces provide the same capabilities, i.e, the ability to specify the topology of a model and to define and edit parameters associated with each object.

The Graphical User Interface and Dialogue Editor (GUIDE) allows users to assemble EQN models graphically from elements of a set of icons representing the various object types, interconnected by routing paths. Users can define or edit object-specific parameters through the use of "dialogue" windows.

GUIDE runs on a 512K Macintosh computer. It resembles a typical application program on the Macintosh in its use of the window environment, mouse input, and pull-down menus. GUIDE runs standalone; no host computer is needed during the specification process. This reduces the load on the mainframe that executes the simulation. GUIDE produces a file containing the EQN model specifications, which is transferred to the host for processing by the translator.

The **Textual Interface and Dialogue Editor (TIDE)** is provided as an alternative when a Macintosh is unavailable. It runs on a Unix-based system and is intended to be used with a VT100 or equivalent terminal. TIDE uses the CURSES windowing library routines[21] for efficient screen management.

The user is guided through a series of hierarchically structured menus. A unique feature of this interface is that, unlike the dialogue interfaces for some performance evaluation tools, interaction is menu-driven and conducted via overlapping "windows" on the screen. This gives the user an improved sense of perspective about the current context of the specification process. The output of TIDE is also a specifications file to be processed by the translator.

3. GIST Objects

A GIST model is constructed of a number of objects, each object of a predefined object type. Jobs are modeled by "job-processes" in CSIM. A visit by a job to an object is modeled by the corresponding job-process invoking the CSIM library object routine associated with that object. Each job belongs to a **jobclass**, and activity at the various object types is often jobclass-dependent. A list of GIST object types, organized based on the modeling capabilities that the object types provide, is presented below.

3.1. Management of active resources

Queueing network models are typically used to evaluate performance of systems with a limited number of service facilities. A service facility is, in abstract, a resource for which jobs may contend -- a resource that is actively engaged in providing a service, and is termed an **active resource**. A job can be assigned only one active resource at a time. Active resources are modeled by **SERVER** or **QSERVER** objects.

Contention between jobs for entry to SERVER objects is modeled by **QUEUE** objects. GIST differs from other simulation packages in that it allows the separation of the two activities of waiting for and receiving service with the two object types QUEUE and SERVER. With certain restrictions, it permits interconnections of a number of QUEUEs with a number of SERVERs to allow straightforward and intuitive specification of complex systems. When this flexibility is not needed, active resource contention can be modeled with QSERVER objects.

QUEUE, SERVER

SERVER objects represent service facilities (CPUs, I/O channels, controllers, etc.) in a system. QUEUE objects model contention for these service facilities. Jobs join a QUEUE object, and wait there until assigned a SERVER object. Jobs at a QUEUE object conform to that object's queueing discipline, a policy which determines the order in which jobs become eligible to leave the object.

User-specified **selection rules** control the routing of jobs between QUEUE and SERVER objects. The selection rules specify (a) how a QUEUE object selects one of a number of SERVER objects to which it can route a job, and (b) how a SERVER object selects one of a number of QUEUE objects that can route a job to the SERVER object. Selection rules are probability-based or priority-based. If priority-based selection rules are specified for both a QUEUE and any of its associated SERVERs, preference conflicts are possible. GIST uses a stable-pair resolution scheme assign SERVERS to QUEUES when this occurs.

GIST presently supports four queueing disciplines: First Come First Served (FCFS), Last Come First Served (LCFS), Last Come First Served with Preemptive Resume (LCFSPR), and Processor-Sharing (PS). Within FCFS, LCFS, and LCFSPR, it is possible to define a jobclass-based priority scheme. GIST also allows priority-based preemption, but at the expense of considerable computational overhead.

Queue length and waiting time statistics are available at QUEUE objects; SERVER objects can collect utilization statistics.

QSERVER

A QSERVER object in GIST is equivalent to a queue in a traditional queueing network model. It is an object where both the queueing and service functions are performed. QSERVERS are computationally more economical than separate QUEUE and SERVER objects. In the current implementation, a QSERVER object has an FCFS queueing discipline. Other common disciplines will be added in the future. The user specifies one of several service time distribution functions. A QSERVER object can collect statistics on queue length, waiting time, and utilization.

3.2. Management of passive resources

In modeling computer systems, we frequently need to describe the ability of a job (a task or process) to acquire some types of resources and continue to hold them even when new resources are acquired. An example is the need for a process to acquire a partition of primary memory before it can receive service by a CPU. These kinds of resources, called **passive resources**, cannot be described by SERVER or QSERVER objects. Acquisition and release of passive resources takes place at **ALLOCATE** and **DEALLOCATE** objects, respectively. **CREATE** and **DESTROY** objects can be used to vary the total quantity of a non-conservative resource.

ALLOCATE

An ALLOCATE object assigns units of a specific passive resource. Jobs request resources in integer amounts according to jobclass-dependent distributions. When a request from a job cannot be satisfied immediately due to an insufficient amount of the resource, the job is delayed until the request is serviced. An ALLOCATE object is characterized by an allocation policy and a resource request distribution for each jobclass. The allocation policy may be First Come First Served (FCFS), First Fit, or FCFS with a jobclass-based priority. Statistics on the number of waiting jobs and waiting time for each jobclass and the quantity of resource

allocated are available at an ALLOCATE object.

DEALLOCATE

All units of a specific resource held by visiting jobs are reclaimed and returned to the pool of available resources. A DEALLOCATE object can collect statistics on the quantity of resource released.

CREATE

When jobs visit a CREATE object, an integer amount of a particular passive resource is produced. The amount of resource created is in accordance with a user-defined distribution. A CREATE object can collect statistics on the quantity of resource created.

DESTROY

All the units of a specific resource held by visiting jobs are deallocated, but they are not returned to the pool of available resources. A DESTROY object can collect statistics on the quantity of resource destroyed.

3.3. Arrivals and departures in open networks

Another essential feature for a computer system evaluation tool is the ability to model the arrival and departure of jobs. EQN models incorporating this feature are called open networks. Jobs are spontaneously generated ("arrive" from an external source), and may subsequently be destroyed ("depart"). Mechanisms necessary for simulation of an open network model are available through **SOURCE** and **SINK** objects.

SOURCE

A SOURCE object creates jobs of a specific jobclass. The creation of jobs can depend on a user-defined condition, or jobs can be created at intervals determined by an inter-generation time distribution specified by the user. A SOURCE object can collect statistics on the number of jobs created and the inter-generation time.

SINK

A job reaching a SINK object is removed from the network. A SINK object does not reclaim any passive resources that may have been committed to a job in the course of simulation. A job is expected to explicitly deallocate or destroy any passive resource it holds, before entering a SINK.

3.4. Concurrency and Synchronization

Two object types, **FORK** and **JOIN**, can be used to model concurrent job activities, with or without synchronization. When a job visits a FORK object, a new job is created. If the created job is specified to be a **child** of the creator job, then a JOIN object can be used to merge the two jobs at a later time, resulting in the termination of the child job. This feature permits representation of nested synchronization. It is also possible to model parallelism without synchronization by creating **peer** jobs at FORK objects.

FORK

A FORK object creates a new job when a job of a specific jobclass visits it. The new job is defined to be either a child or a peer job in relation to the visiting (creator) job. The created job can have a jobclass different from that of the creator job.

A peer job is completely independent, with no relation between it and the creator job. A child job may eventually meet its creator or parent job at a JOIN object and terminate. It is possible, furthermore, for the new job to visit a FORK object and create a child or a peer job. When child jobs create their own child jobs, a hierarchy of jobs results. By employing a corresponding hierarchy of JOIN objects, a user can model nested synchronization. A FORK object makes available statistics on the number of jobs created.

JOIN

When a job with child jobs reaches a JOIN object, it waits there until its most recent non-terminated child job arrives at the same JOIN. If it has a parent, it waits for the parent to arrive and then terminates. When the child job is terminated, the parent job is released to continue through the network. Jobs that have no child or parent jobs are unaffected by visits to JOIN objects and are not delayed. Statistics on waiting time can be collected at a JOIN object.

3.5. Routing specification

As jobs are routed through a computer system, decisions must be made as to the order in which they visit the various parts of the system. One approach to incorporating routing policies into an EQN model is to make the routing policy a part of the specification for each object. GIST uses an alternative approach, which is to allow most object types to have at most one output. Route selections are performed by **SWITCH** objects.

SWITCH

GIST allows specification of one **next-object** (the object to which jobs can be routed from the current object) for every object type except QUEUE, SWITCH, and SINK. The next-object can be a SWITCH, where a routing decision is made. The routing decision made at a SWITCH object can be based on a condition evaluation, a random selection, or a static, jobclass-based routing specification.

In a condition-based routing policy, each choice of next-object has an associated boolean expression, which, if evaluates to TRUE, will cause that choice to be taken. The expressions are evaluated in a specified order. Also, jobs are routed to a "default" next-object if none of the expressions evaluate to TRUE. For random selections, each choice of next-object has an associated probability. In the jobclass-based policy, for each jobclass, there is exactly one next-object to which jobs can be routed. It is

possible to perform complex job routing by cascading SWITCH objects. A SWITCH object does not delay the jobs that visit it. Statistics on the number of jobs routed to each next-object can be collected at a SWITCH object.

3.6. Network-wide statistics collection

Statistics may relate to actions at a specific object or to global behavior. Mechanisms for collecting object-specific statistics are available for most object types. When the statistics of interest are not local to a single object, the object type **PROBE** can be used to collect data on specific jobs or jobclasses.

PROBE

A PROBE object collects user-specified statistics from visiting jobs. Jobs are unaffected by visits to PROBE objects. The statistics that can be collected at a PROBE, on a jobclass basis, are inter-arrival times, number of arrivals, transit time between an object and the PROBE object, and number of visits by a job to a specific object between visits to the PROBE object.

4. GUIDE

GUIDE is written in the C language, using the Stanford University Mac C (sumacc) development software[22]. It is tailored to the Macintosh and presently is not portable to other computers/graphics terminals. Users familiar with typical Macintosh application programs can learn to use GUIDE in a few minutes.

The software for GUIDE can be divided into two parts, the graphical input package and a set of specification routines. The input package allows the user to create a graphical representation of the network topology, i.e., the objects and their interconnections. It also handles all generic system functions such as saving partial or complete specification files, and opening existing files. The specification routines allow the object-specific data for individual objects to be entered through windows tailored to each object type.

GUIDE presents the user with a set of menus, a working window and an options window (Fig. 2). The working window is the region of the screen displaying the part of the larger work area in which the user is currently creating the model. The work area can be scrolled horizontally and vertically in the working window to help in specifying large models. The options window contains graphical representations (icons) for each object type available in GIST, as well as an interconnect option.

There are seven pull-down menus. The apple menu is found in other most Macintosh applications and is not directly relevant to the interface, but has been provided to maintain consistency. The file menu lets a user open a new file, open an existing file, close the current file, save the current specifications in a file, save the current specifications in a different file, revert back to the last saved version of the specifications, check for partial correctness of the specifications, and quit the program.

The edit menu allows a user to delete objects or interconnections. It also permits items to be cut, copied, or pasted, although these features presently work only with concurrent mini-applications ("desk accessories") that are invoked via the apple menu.

The help menu (currently unimplemented) is used to invoke on-line help. The specify menu lets users specify initial jobs, run-time parameters, and initial quantities of passive resources. In addition, it lets the user view information about job classes and objects. The transfer menu permits the user to quit GUIDE and start another application program relatively quickly. The debug menu is used only for debugging the interface software. It contains items for displaying error messages and internal data structures.

5. An Example of Model Specification Using GUIDE

Consider a computer system where jobs arrive from the external world, receive service from the CPU, and depart. This system can be modeled using the GIST objects SOURCE, QSERVER and SINK. The SOURCE object models the arrival of new jobs and the QSERVER models the CPU. The SINK object models the departure of jobs from the system by removing them from the network. A PROBE object is used between the QSERVER and the SINK to collect statistics.

GUIDE allows an object type to be selected by positioning the curser over the appropriate icon in the options window and clicking (a single push/release of the mouse button). The user creates a new instance of an object type by dragging the selected object icon to the desired position within the working window.

Interconnecting two objects in the working window is accomplished using the interconnect option in the options window. The user clicks first on the source icon and then on the destination icon. During this process an "elastic" line anchored at the source object tracks the motion of the cursor. Interconnections with more than one line segment can also be specified by clicking in succession on the source icon, intermediate points that define the line segments, and the destination icon. The completed EQN model diagram for this example appears in Fig. 3.

The specifications for each object are entered through a dialogue window. Double clicking an object icon causes the dialogue window to appear. For example, double clicking on the SOURCE icon brings up the SOURCE specification dialogue window in Fig. 4. This window contains text boxes for specifying the name of the object and the class of jobs that are created by this SOURCE, and, optionally, boxes for distribution parameters. Other items in the dialogue window include "check" items to specify the statistics that can be collected at this SOURCE, and OK and CANCEL buttons. Clicking on the OK button confirms the specifications given for the object and causes the dialogue window to disappear. The effect of clicking the CANCEL button is to undo all changes made to the object's specification and return to the working window. The QSERVER and PROBE objects can be similarly specified. No dialogue window is necessary for the SINK object.

The user enters run-time parameters such as the length of the simulation (in simulated time) through another dialogue window which is invoked by selecting the "Run Parameters" item in the specify menu. We can also specify at this point that the simulation is to produce a trace of events, by clicking on the Event trace check box in that dialogue window. The model specification is now complete and can be saved in a file to be later transmitted to the host computer for processing by the translator.

6. TIDE

TIDE accepts model specifications on character-oriented terminals[23]. It offers to the user a consistent set of mechanisms that are collectively easy and intuitive to use. The user can add new objects or delete previously defined objects and can view or edit the specification of attributes of any object in the model. Attribute specification can be done in any desired order, and partial specifications from an editing session can be saved. The available alternatives at any point in an editing session appear as a menu. Specification of a list of properties is frequently performed in a sequence of overlapped windows on the screen. Each of these overlapped windows appears as a part of the window within which specification of the object attributes begins. The textual interface is written using a screen updating and cursor movement optimization package, available as the CURSES library on Unix, making the TIDE implementation terminal independent.

Fig. 5 presents a sample view of the terminal screen during an editing process. An ALLOCATE object is being specified, and at the instant shown, a menu for "ALLOCATION POLICY" allows the user to select one of two policies, FCFS and FIRST FIT. The user can discontinue the specification of an object by scrolling the curser, up or down, until it is past the first or the last lines in the list of attributes. A new object can be added, or the specifications for a previously defined object can be edited, by selecting the appropriate option from the menu which appears at that point.

7. Examples of GIST Models

We present two examples of the application of GIST to performance evaluation of computer systems. The first example illustrates the capabilities to model (a) condition-based process creation, (b) routing based on different policies, and (c) contention for passive resources. The second example demonstrates the utility of separation of QUEUE and SERVER objects.

7.1. 1. Memory management

Consider a computer system in which processes dynamically acquire memory. Specifically, each process goes through the following cycle. (1) After activation, the process acquires a certain amount of memory. (2) It is then placed on a list of ready-to-run processes, and awaits execution at the CPU. (3) After receiving service at the CPU, the process may either terminate, or it may request more memory from the system before execution can continue. We assume that a process, upon termination, does not explicitly relinquish the memory it has acquired from the system. (4) The system activates a garbage collection process whenever the available amount of memory falls below a certain critical value. The garbage collector is assigned a high execution priority over other processes at the CPU. Some of the performance measures of interest may be the amount of time a process spends in the system, or the frequency with which the garbage collector is invoked.

The EQN model for this system is shown in Fig. 6. The details of the various objects are as follows.

(1) **S1** and **S2** are SOURCE objects. **S1** creates the jobs that model user or system processes. Generation of jobs at **S1** is governed by an intergeneration-time distribution function. **S2** generates the garbage collection processes, and is condition-based. **S2** also collects the statistics on intergeneration time, from which the frequency of garbage collection process can be determined. The GIST state variables used in formulating the condition under which a garbage collection process is created at **S2** are the statistics collected at the PROBE and CREATE objects in the model, as explained below.

(2) **A1** and **A2** are ALLOCATE objects. Acquisition of memory by processes is modeled by allocation of a passive resource at these objects.

(3) **D** is a DESTROY object. Since a process does not explicitly relinquish the memory it holds when it terminates, the system "loses" this memory. This loss is modeled by the DESTROY object. While it is true that the total amount of memory in the system does not vary, and hence that it is a conservative resource, we find it convenient to model the temporary unavailability of unused memory by removing it from the system. Analogously, reclamation of the resource can be modeled by a CREATE object, **C**, that the garbage collector visits after leaving **S2**.

(4) **CPUQ** is a QUEUE object, with two priority classes. Jobs from **S2** join the higher priority class at **CPUQ**. **CPU** is a SERVER object. **SW1** and **SW2** are SWITCH objects. **SW1** performs a jobclass-based routing to send the garbage collector jobs to the PROBE **P4**. **SW2** performs a probability-based routing, so that some of the jobs may return to **CPUQ** for further execution.

(5) **P1**, **P2**, **P3**, and **P4** are PROBE objects. **P1** counts the number of jobs leaving **A1**. **P2** counts the number of jobs routed back to the CPU queue, **CPUQ**. The ALLOCATE objects collect statistics on the resource allocation to the visiting jobs. **P3** counts the number of jobs leaving the system along the path through the DESTROY object **D**. **P3** also measures the lifetimes of the visiting jobs. **D** collects statistics on the amount of resource destroyed. **P4**, likewise, monitors the garbage collector jobs that are routed to it at the SWITCH **SW1**. The statistic on the amount of resource created at **C** is collected at **C** itself.

It is thus clearly possible to compute the total amount of resource allocated, created, and destroyed in the system, and hence the amount of resource available in the system. This can then be used to specify the condition for the activation of S2.

7.2. 2. Processor availability

In this example, we consider the performance of a shared resource multiprocessor system with attention to reliability concerns. The multiprocessor system consists of a number of heterogeneous processors, each with limited, private resources (such as memory), and sharing some global resources (shared memory and disks). The computation tasks that arrive for execution are organized into one or more groups, with possibly different processor assignment policies followed for each group of tasks. The decision to assign a certain computational task to a particular processor may be based on such considerations as the capabilities of the particular processor, or on the communication overhead of assigning tasks to it. We assume that these factors can be stochastically modeled for each job group by assigning probabilities to the event that a particular processor is selected for the execution of a particular job. Furthermore, the processors are associated with certain reliability characteristics. We can characterize the multiprocessor system with an overall failure rate, and each processor with an individual failure rate.

Fig. 7 shows an EQN model representation of this system. The QUEUE objects **JOBQ1** and **JOBQ2** represent groups of jobs. SERVER objects **SERV1**, **SERV2**, **SERV3** represent the processors. Jobs of a special type model failures, and these jobs join a third QUEUE object, **FAILQ**. Jobs in the **FAILQ** object select each of the SERVER objects with certain probabilities. The SERVER objects select input QUEUE objects in a priority-based manner, with the **FAILQ** selected with a higher priority than the other two. JOB1Q and JOB2Q have the same priority at all the SERVER objects, and are selected equiprobably. The jobs from all the SERVER objects are routed via a SWITCH to a PROBE object that measures the throughputs over all the jobs, except for those from the **FAILQ** QUEUE object.

Since the jobs from **FAILQ** have the highest priority in server assignment, they make the servers unavailable to the other jobs in the system. They in effect model the unavailability of a failed processor. It is possible to vary the probabilities used in selection of SERVER objects by jobs from the two QUEUE objects JOB1Q and JOB2Q, and to then study the resulting effect on the overall throughput of the multiprocessor.

8. Future Directions

GIST is continually evolving in an effort to enhance its capabilities and to remove some of its limitations. Experience with the present version of GIST has shown the need for additional modeling capabilities. Planned GIST extensions include the addition of several new object types, modification of existing object types, and provision of job variables and global variables.

From a user's point of view the only accessible information that a job carries with it is its class and its relationship, if any, to other jobs. **Job variables** allow jobs to carry additional numerical information that can be used in modeling systems where a job maintains a state that is a function of its history. Objects can perform actions based on the values of job variables. A new object type ASSIGN will allow the values of job variables to be changed.

Explicit state information could be maintained for the entire model in the form of **global variables**. As in the case of job variables, permitting actions at objects to be based on global variables which can be modified by ASSIGNs would result in greater modeling flexibility.

New object types being considered are ASSIGN, WAIT, WAIT-SEMAPHORE, SIGNAL-SEMAPHORE, and DELAY. The ASSIGN object is discussed above. The WAIT object makes jobs wait on the values of conditions. The WAIT-SEMAPHORE and SIGNAL-SEMAPHORE objects directly implement the semantics of P and V operations on a binary semaphore. A DELAY object delays a job for an amount of time drawn from a probability distribution, independently of any other jobs that might be delayed at the same object.

Modifications to existing object types being considered include: additional queueing disciplines at a QSERVER; jobclass-based statistics collection at QUEUE and SERVER; jobclass-based service time distributions at a SERVER; generalized priority queues and additional allocation policies at an ALLOC; variable amounts of resource to be deallocated or destroyed at a DEALLOC or DESTROY; creation of multiple peer or child jobs by a single job at a FORK; and job variable-based routing at a SWITCH.

The three most significant extensions planned for GIST are (1) a SUBMODEL object type; (2) a USER object type; and (3) statistics analysis. The SUBMODEL object type is a means of encapsulating other object types into a single object to allow the user to construct hierarchical models. This allows details to be hidden when they are not necessary, making it easier to create and understand large, complex models. For instance, a submodel composed of QUEUEs and SERVERs to model a uniprocessor could be used in building the model of a multiprocessor system. The SUBMODEL object will also permit users to have libraries of previously defined submodels that can be included in new model specifications.

The USER object is an escape mechanism for describing system features that cannot be adequately modeled with the predefined object types. A USER object semantics will be determined by a user-supplied C procedure that is invoked by a job-process when the job visits the object. This runs counter to the GIST philosophy of using high level abstract object types since it requires that the user know many implementation details of how processes invoke procedures and how the CSIM data structures are used by the procedures.

At present, no provision exists in GIST for the analysis and validation of simulation results. Also, simulations are terminated either on the basis of simulation time or a user-defined condition, and not on the basis of statistics-based stopping rules. We intend to include facilities

for confidence interval estimation and sequential stopping rules based on these estimates.

REFERENCES

[1] R.G. Covington, "CSIM: An Efficient Implementation of a Discrete Event Simulator," M.S. Thesis, Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892 (April 1985).

[2] C.H. Sauer, E.A. MacNair, and J.F. Kurose, "The Research Queueing Package Version 2: Introduction and Examples," RA 138, IBM T.J. Watson Research Center, Yorktown Heights, NY (April 1982).

[3] W. Whitt, "The Queueing Network Analyzer," BSTJ **62**(9, Part 1), pp.2779-2815 (November 1983).

[4] W. Whitt, "Performance of the Queueing Network Analyzer," BSTJ **62**(9, Part 1), pp.2817-2843 (November 1983).

[5] A.K. Agrawala, S.K. Tripathi, M. Abrams, K.K. Ramakrishnan, M. Singhal, and S.H. Son, "STEP-1: A User Friendly Performance Analysis Tool," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[6] M. Veran and D. Potier, "QNAP2: A Portable Environment for Queueing Systems Modelling," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[7] H. Beilner and J. Maeter, "COPE: Past, Presence and Future," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[8] A. Brandwajn, "Issues in Mainframe System Modelling - Lessons from Model Development at Amdahl," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[9] M. Booyens, P.S. Kritzinger, A. Krzesinski, P. Teunissen, and S. van Wyk, "SNAP: An Analytic Multiclass Queueing Network Analyser," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[10] S.C. Bruell, G. Balbo, S. Ghanta, and P.V. Afshari, "A Mean Value Analysis Based Package for the Solution of Product-Form Queueing Network Models," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[11] K.G. Ramakrishnan and D. Mitra, "An Overview of PANACEA, a Software Package for Analyzing Markovian Queueing Networks," BSTJ **61**(10, Part 1), pp.2849-2872 (December 1982).

[12] I. Kino and S. Morita, "PERFORMS - A Support System for Computer System Performance Evaluation," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[13] J. Buzen, "BEST/1 - Design of a Tool for Computer System Capacity Planning," Proc. 1978 AFIPS National Computer Conference **47**, pp.447-455 (1978).

[14] B. Mueller, "NUMAS: A Tool for the Numerical Modelling of Computer Systems," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[15] PAWS - Performance Analyst's Workbench System: INTRODUCTION AND TECHNICAL SUMMARY, Information Research Associates, Austin, TX (July 1983).

[16] T. Nishida, M. Murata, H. Miyahara, and K. Takashima, "PLANS: Modelling and Simulation System for LAN," Proc. International Conference on Modelling Techniques and Tools for Performance Analysis, Paris (May 1984).

[17] B. Melamed and R.J.T. Morris, "Visual Simulation: The Performance Analysis Workstation," Computer, (to appear) (1985).

[18] A. Blum, E.A. MacNair, and C.H. Sauer, "The Research Queueing Package: Graphics Developments," RC 9513, IBM T.J. Watson Research Center, Yorktown Heights, NY (August 1982).

[19] J.C. Browne, D. Neuse, J. Dutton, and K.-C. Yu, "Graphical Programming for Simulation of Computer Systems," Proc. 18th Annual Simulation Symposium, pp.109-126 (March 1985).

[20] J.B. Sinclair, K.A. Doshi, and S. Madala, "GIST: The Graphical Input Simulation Tool," TR 8511, Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892 (May 1985).

[21] K.C. Arnold, "Screen Updating and Cursor Movement Optimization: A Library Package," Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

[22] S. Madala, "Design of a Graphical Input Simulation Tool for Extended Queueing Network Models," M.S. Thesis, Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892 (April 1985).

[23] K.A. Doshi, "Extended Queueing Network Modeling," M.S. Thesis, Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892 (April 1985).

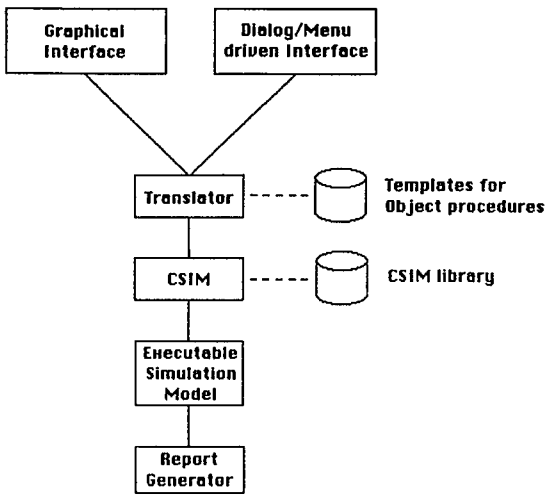

Figure 1. Organization and Components of GIST

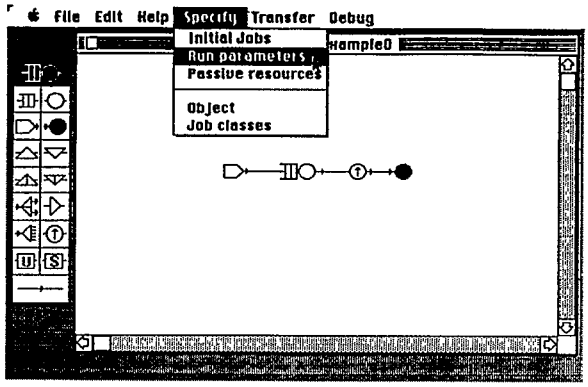

Figure 3. Model diagram.

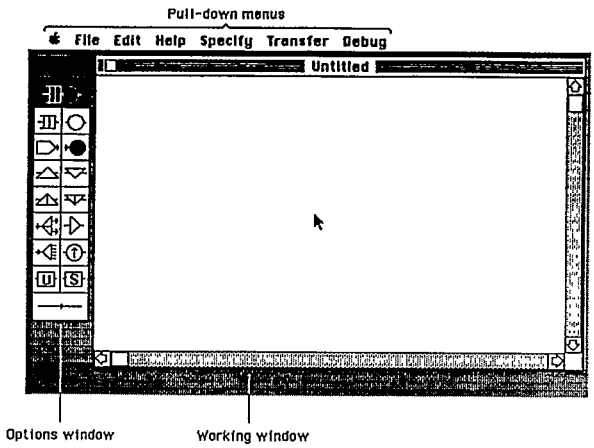

Figure 2. GUIDE windows and menus.

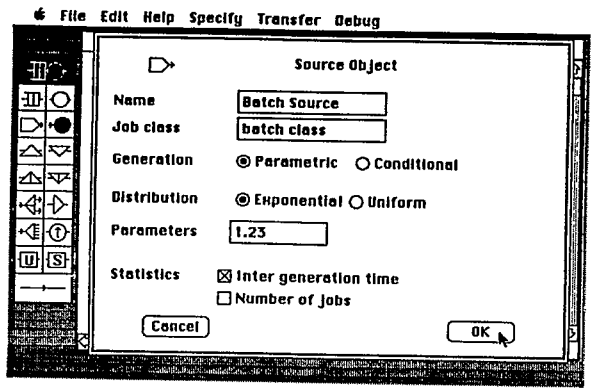

Figure 4. Dialogue window for SOURCE object.

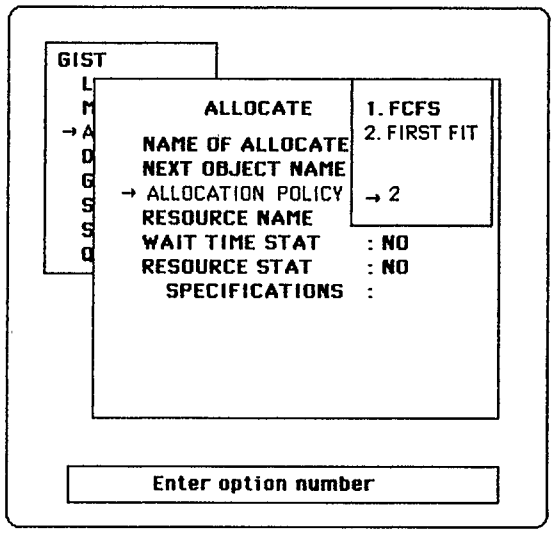

Figure 5. TIDE: Menu-guided specification

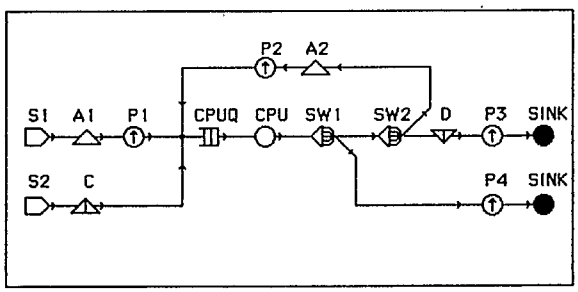

Figure 6. GIST model for example 1.

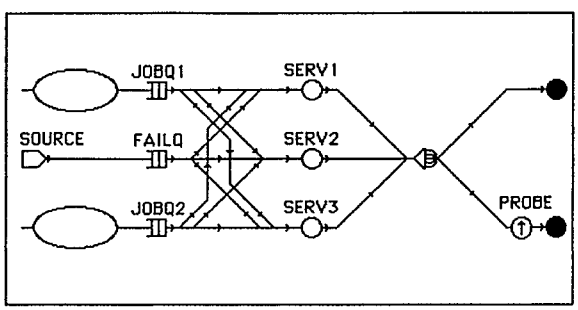

Figure 7. GIST model for example 2.

James B. Sinclair
J. B. Sinclair is an associate professor in the Department of Electrical and Computer Engineering at Rice University. He received his Ph.D. in 1978 from Rice. His research interests are computer architecture, computer networks, distributed systems, and performance evaluation. He is a member of the the ACM, IEEE, and the IEEE Computer Society.

Department of Electrical and Computer Engineering
Rice University Houston, TX 77521-1892

Kshitij A. Doshi
K. Doshi is a graduate student in the department of Electrical and Computer Engineering at Rice University. He received his M.S degree in 1985 from Rice and his B.Tech degree in 1982 from the Indian Institute of Technology, Bombay, India. His research interests are in parallel processing, architecture, and performance evaluation. He is a member of the IEEE.

Department of Electrical and Computer Engineering
Rice University Houston, TX 77521-1892

Sridhar Madala
S. Madala is a graduate student in the department of Electrical and Computer Engineering at Rice University. He received his M.S in 1985 from Rice and his B.Tech in Electrical Engineering in 1982 from the Indian Institute of Technology, Madras, India. His research interests are in distributed systems and simulation.

Department of Electrical and Computer Engineering
Rice University Houston, TX 77521-1892