DESIGN OF A GRAPHICAL INPUT SIMULATION TOOL FOR EXTENDED QUEUEING NETWORK MODELS

by

SRIDHAR MADALA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE:

James B. Sinclair, Associate Professor of Electrical and Computer Engineering, Director

Faye A. Briggs, Associate Professor of Electrical and Computer Engineering

Peter J. Varman, Assistant Professor of Electrical and Computer Engineering

Houston, Texas
April, 1985
DESIGN OF A GRAPHICAL INPUT SIMULATION TOOL FOR
EXTENDED QUEUEING NETWORK MODELS

by

Sridhar Madala

ABSTRACT

In analyzing the performance of computer and digital communication systems, contention for finite capacity resources is often seen to be a dominant factor. Extended Queueing Network (EQN) models are appropriate for modeling such systems and have been used with considerable success. EQN models can be solved by exact analysis, by approximate analysis, or by simulation.

This thesis is concerned with the design of a high level tool for solving EQN models by simulation that accepts specifications in a natural graphical manner. The primary motivation for this research was to prove the feasibility of providing a versatile tool that is easy to learn and use and is complete with respect to EQN models. Existing software tools for EQN modeling do not take advantage of the fact that a natural way to specify such models is graphical.

Our modeling tool, the Graphical Input Simulation Tool (GIST), achieves these objectives: 1) by utilizing a transaction-oriented approach as opposed to a language-based approach, 2) by providing two
user interfaces, a graphical interface and a textual interface, that permit specification of EQN models at a very high level of abstraction, and 3) by means of a versatile set of modeling abstractions. In terms of modeling capabilities, GIST provides analogs of most abstractions commonly found in other high-level EQN modeling tools and also includes abstractions that have no counterparts in other tools.

We demonstrate the feasibility and utility of providing a GIST-like tool and conclude that the transaction oriented-approach is also applicable and appropriate for building modeling tools in areas other than EQNs. GIST can also be extended to further research in performance evaluation tools and in performance evaluation in general.
Acknowledgements

I would like to thank Bart Sinclair for making this research possible and especially for his patient editing of this thesis. I would also like to thank Dr. Faye Briggs and Dr. Peter Varman for being on my thesis committee and for their many helpful suggestions.

Working with Kshitij Doshi has been a rewarding experience and I thank him for sharing his ideas with me. This work would have taken a lot longer without Richard Covington's help and I thank him for his many patient answers to my numerous questions about CSIM, and Unix.
CONTENTS

1 Introduction ................................................................. 1
  1.1 Performance Evaluation and Modeling .............................. 1
  1.2 EQN Models ............................................................. 2
  1.3 Thesis Goals .......................................................... 3
  1.4 Thesis Organization .................................................. 4

2 EQN Models and Performance Evaluation Tools ....................... 5
  2.1 Introduction .......................................................... 5
  2.2 EQN Models ........................................................... 5
  2.3 Performance Evaluation Tools ...................................... 7
  2.4 Thesis Goals ......................................................... 10
  2.5 Summary ............................................................... 11

3 Overview of GIST ........................................................ 12
  3.1 Introduction .......................................................... 12
  3.2 Discrete Event Simulation and CSIM .............................. 14
  3.3 Object Routines ...................................................... 15
  3.4 User Interface ....................................................... 17
    3.4.1 Graphical Interface ........................................... 17
    3.4.2 Textual Interface ............................................... 18
  3.5 Translator ........................................................... 18
  3.6 Summary ............................................................... 19

4 GIST Modeling Capabilities ............................................ 20
  4.1 Introduction .......................................................... 20
  4.2 EQN Elements ........................................................ 21
    4.2.1 Jobs and job classes .......................................... 21
    4.2.2 Active and Passive resources ............................... 22
    4.2.3 Statistics Collection ....................................... 23
    4.2.4 Conditions .................................................... 24
    4.2.5 Routing ........................................................ 25
    4.2.6 GIST object types ............................................ 25
  4.3 Active Resource Modeling ......................................... 27
    4.3.1 QUEUE object ................................................. 29
      4.3.1.1 Queueing Discipline ....................................... 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.1.2 Server Selection</td>
<td>31</td>
</tr>
<tr>
<td>4.3.2 SERVER object</td>
<td>32</td>
</tr>
<tr>
<td>4.3.2.1 Service time distribution function</td>
<td>32</td>
</tr>
<tr>
<td>4.3.2.2 Queue Selection Rule</td>
<td>32</td>
</tr>
<tr>
<td>4.3.3 Restrictions on Queueing Disciplines and Server Selection Rules</td>
<td>33</td>
</tr>
<tr>
<td>4.3.4 QSERVER object</td>
<td>34</td>
</tr>
<tr>
<td>4.4 Open systems modeling</td>
<td>35</td>
</tr>
<tr>
<td>4.4.1 SOURCE object</td>
<td>36</td>
</tr>
<tr>
<td>4.4.2 Sink object</td>
<td>36</td>
</tr>
<tr>
<td>4.5 Passive resource modeling</td>
<td>37</td>
</tr>
<tr>
<td>4.5.1 ALLOC object</td>
<td>37</td>
</tr>
<tr>
<td>4.5.2 DEALLOC object</td>
<td>39</td>
</tr>
<tr>
<td>4.5.3 CREATE object</td>
<td>39</td>
</tr>
<tr>
<td>4.5.4 DESTROY object</td>
<td>40</td>
</tr>
<tr>
<td>4.6 Modeling Concurrency and Synchronization</td>
<td>40</td>
</tr>
<tr>
<td>4.6.1 FORK object</td>
<td>40</td>
</tr>
<tr>
<td>4.6.2 JOIN object</td>
<td>41</td>
</tr>
<tr>
<td>4.6.3 Usage of FORK and JOIN</td>
<td>42</td>
</tr>
<tr>
<td>4.7 Routing and statistics collection</td>
<td>43</td>
</tr>
<tr>
<td>4.7.1 Switch object</td>
<td>43</td>
</tr>
<tr>
<td>4.7.1.1 Probability-based Routing</td>
<td>43</td>
</tr>
<tr>
<td>4.7.1.2 Class-based Routing</td>
<td>45</td>
</tr>
<tr>
<td>4.7.1.3 Condition-based Routing</td>
<td>45</td>
</tr>
<tr>
<td>4.7.2 PROBE object</td>
<td>45</td>
</tr>
<tr>
<td>4.8 Summary</td>
<td>47</td>
</tr>
<tr>
<td>5 Graphical Interface</td>
<td>48</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>48</td>
</tr>
<tr>
<td>5.2 Overview of the Graphical Interface</td>
<td>50</td>
</tr>
<tr>
<td>5.3 Design of the Interface</td>
<td>50</td>
</tr>
<tr>
<td>5.3.1 Hardware Requirements for Interface Implementation</td>
<td>51</td>
</tr>
<tr>
<td>5.3.2 Implementation Overview</td>
<td>51</td>
</tr>
<tr>
<td>5.3.3 Interface Description</td>
<td>52</td>
</tr>
<tr>
<td>5.4 Example Usage</td>
<td>56</td>
</tr>
<tr>
<td>5.5 Summary</td>
<td>64</td>
</tr>
<tr>
<td>6 Modeling with GIST</td>
<td>67</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>67</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>6.2</td>
<td>Modeling contention for passive resources</td>
</tr>
<tr>
<td>6.3</td>
<td>Nested synchronization</td>
</tr>
<tr>
<td>6.4</td>
<td>Separation of Waiting and Service functions</td>
</tr>
<tr>
<td>6.5</td>
<td>Summary</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
</tr>
<tr>
<td>7.1</td>
<td>Status Report</td>
</tr>
<tr>
<td>7.2</td>
<td>An Evaluation of the Transaction-Oriented Approach</td>
</tr>
<tr>
<td>7.3</td>
<td>An Evaluation of GIST</td>
</tr>
<tr>
<td>7.4</td>
<td>Applications of GIST</td>
</tr>
<tr>
<td>7.5</td>
<td>Future Directions</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Additional capabilities</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Enhancements to the user interface</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Alternative methods of solution</td>
</tr>
<tr>
<td>7.5.4</td>
<td>Application to other tools</td>
</tr>
<tr>
<td>7.6</td>
<td>Conclusion</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
</tr>
</tbody>
</table>
Introduction

1. Performance Evaluation and Modeling

Evaluation of a system's performance is an essential aspect of any engineering activity. The performance of a system is of concern to various groups of people during all stages of its lifetime. Engineering systems are evaluated by their designers, manufacturers, buyers, managers, and users. The performance of a system can be represented by quantifiable indices which are given different weights by different people and even by the same people under different circumstances. Performance evaluation, then, is the characterization of the behavior of a system under different conditions.

A system's performance can be evaluated in various ways: it can be measured, calculated or estimated. Direct measurement of the parameters of interest, though highly accurate, is not always suitable. It necessitates the existence of a prototype, often an expensive proposition. Even if a prototype does exist, all parameters might not be directly measurable or the measurement might not be economically viable.

Alternatively, performance information can be obtained from a model of the system. A model is an abstract representation of the system which consists of certain organized information about it. Considerable skill is required in modeling a complex system in order to include detail that is sufficient to obtain results to the desired
accuracy. Irrelevant detail should be excluded so as to make the solution of the model less difficult.

Models can be broadly classified as being deterministic or probabilistic. Turing machines, finite state automata, graph models of programs and Petri nets all fall into the former category. Deterministic models are limited in scope and can usually be used only to estimate order of magnitude or bounds of certain parameters. Probabilistic models, on the other hand, are well suited for studies of systems such as those with contention for critical resources. Markov, semi-Markov, and Queueing Network (QN) models belong to this class.

2. EQN Models

QN models have been widely used with considerable success in the study of computer systems. QN models are primarily composed of two abstractions: queues and servers. Servers represent entities which provide service to tasks or jobs, usually one job at a time and queues represent waiting lines where jobs wait to receive service. Generalized or Extended Queueing Networks (EQNs) permit modeling of a broader class of systems. Classes of systems that can be modeled using EQNs but not QNs include those where jobs are permitted to hold more than one type of resource simultaneously and those which permit synchronization between jobs.

EQN models can be solved analytically, numerically, or by simulation. Analytical techniques lead to exact solutions and are pre-
ferred over the other methods. These techniques are applicable to only a small subset of all possible EQN models. Simplifying assumptions can sometimes be made to analytically intractable models to enable numerical solutions. As models become increasingly complex, numerical techniques can lead to results with unacceptable accuracy. When models are mathematically intractable or are not amenable to numerical techniques one recourse is to execute a simulation model to mimic the behavior of the system under study.

3. Thesis Goals

Several performance evaluation tools allow users to specify and solve EQN models using one or more of the above mentioned techniques. The available tools range from extensions of conventional programming languages to special purpose high level tools that provide representations of common components of an EQN model.

Most existing EQN modeling tools require the user to be conversant with either a programming language or details of usage. Often a user translates a graphical representation of the model into input specifications appropriate to the tool. This manual translation process is time consuming and a potential source of errors in the in the model specification process. Also, the user is often more concerned with details of how to use the tool than with the model itself. These tools fail to make use of the fact that the most intuitive manner in which to specify an EQN model is at least partially graphical.
The Graphical Input Simulation Tool (GIST) is a simulation-based performance evaluation tool for specifying and solving EQN models. Our aim in designing and implementing GIST is to prove the feasibility of providing users with a simple, intuitive, easy-to-learn and easy-to-use tool that allows specification of EQN models in a graphical manner while shielding users from implementation details.

4. Thesis Organization

Chapter 2 discusses previous work in the development of performance evaluation tools and defines EQN models, more precisely. This is followed by an overview of GIST in chapter 3, where the major subsystems of GIST and the interactions between them are discussed. Chapter 4 contains a detailed functional description of the various modeling features of GIST, and chapter 5 discusses the graphical user interface of GIST. Some example EQN models solved using GIST are illustrated in chapter 6. Finally chapter 7 contains a summary and some suggestions for future work.
EQN Models and Performance Evaluation Tools

1. Introduction

Generalized or Extended Queueing Networks (EQNs) are well suited for modeling systems where contention for resources is a dominant factor affecting the performance. Examples of such systems include computer systems, and digital communication networks. Considerable work has been done in the past two decades in the development of EQN modeling tools. In this chapter we define EQN models and discuss previous work in the development of performance evaluation tools.

Section 2 contains a definition of QN and BQN models. This is followed by a survey of performance evaluation tools in section 3 with emphasis on tools that permit simulation based solution of BQN models. Section 4 contains a description of the problem and the primary motivation for this research and a summary is included in section 5.

2. EQN Models

Systems in which jobs or tasks place demand on finite capacity resources can be termed queueing systems. Examples of such systems include computer systems where tasks contend for the CPU or an I/O device and digital communication networks in which the communication medium is contended for by various nodes.
Queueing Network (QN) models are composed of the following entities: a set of objects, interconnections between these objects, and a population of jobs that move among the objects. These objects include queues, servers, sources, and sinks. Servers represent entities which provide service to jobs or tasks and queues represent the lines in which jobs wait to receive service. QNs in which the jobs remain within the network are known as closed networks whereas those in which jobs may arrive or depart and the number of jobs may not be constant are known as open systems. Sources represent entities that enable the arrival of jobs into a network and sinks are entities that remove jobs from the network.

Some of the important attributes of a server are, the amount of service given to a job, and the conditions under which it terminates service to a job. Typically, a server can serve only one job at a time. A queue is primarily characterized by its queueing discipline or the rule which determines the order in which waiting jobs become eligible for service. Servers cannot be used to represent certain types of resources which are allocated to a job for a period of time during which the job requests and receives service from other servers. Such resources are termed "passive" resources as they are not actively engaged in providing service to a job.

Generalized or Extended Queueing Networks (EQNs) provide modeling abstractions that permit representation of passive resources. In addition, EQNs provide additional abstractions that permit modeling systems with synchronization between jobs.
Typically, an EQN model is composed of a set of nodes or objects. These include queues, servers, sources and sinks for modeling job entry and exit, objects for modeling allocation, deallocation, creation and destruction of passive resources and objects for modeling job synchronization. Complex job routing patterns are also permitted between objects. EQNs permit the modeling of much more complex systems than is possible with QNs. This additional flexibility is not without its attendant penalty however, since unlike QNs which often have analytical solutions, EQNs are often analytically intractable. Simplifying assumptions can be made in such circumstances, with the resulting loss of accuracy in the system representation, to obtain approximate models which have analytical solutions. An attractive alternative in situations where the model is intractable and simplifying assumptions would lead to results of unacceptable accuracy, is to construct and execute a simulation model to mimic the behavior of the system under consideration.

3. Performance Evaluation Tools

A number of tools for performance evaluation have been reported in the literature. This section contains a survey of previous work on performance evaluation tools with emphasis on tools that permit simulation based solution of EQN models.

Queueing network models have long been used to analyze the performance of computer systems. Several software tools have been
developed over the last two decades to solve EQN models. Most of these solve EQN models by analytic (exact or approximate) or simulation methods and a few permit solution by either. Tools that employ simulation can be classified as either language-based or transaction oriented. Language-based tools typically, are extensions of conventional programming languages to include primitives that support simulation. GASP [1], SIMSCRIPT [2], and CSIM [3] are examples of such tools. Though general purpose language based tools allow developing models with arbitrary complexity, the coding and debugging of a model is still a time consuming and error prone task.

Special purpose tools for solving EQN models have been developed to alleviate these problems. Although, they are less general than language based tools, these special purpose packages result in a faster specification and solution of the model. These tools typically provide a set of abstractions or building blocks which the user can put together to model complex systems.

Perhaps the best known of these is RESQUE, developed at IBM [4]. RESQUE allows numerical as well as simulation methods for model solution. Another unique feature of RESQUE is the availability of three different methods, independent replications, regeneration, and spectral analysis, for confidence interval estimation of simulation results. RESQUE also permits parametric submodels akin to macros in a programming language.

The STEP-1 package developed at the University of Maryland [5] also permits either simulation or analytical methods for model
solution. An interesting feature of STEP-1 is its user interface that operates in one of the three modes 1) a hierarchically structured menu-driven mode; 2) a command mode for the experienced user; and 3) a tutored input mode for the novice. Users can specify one of several computational algorithms in the case of analytical solution.

Examples of other tools that permit simulation as well as analysis include QNAP-2 [6] and COPE [7]. QNAP-2 provides a number of different solution techniques. These include analytical solvers (convolution, mean value analysis, hybrid MVA/convolution algorithms, a markovian solver that generates all model states and transition probabilities and computes steady state probabilities), and discrete event simulation. COPE generates SIMULA67 code in the case of simulation based solution.

Examples of packages that use analytical techniques only include NUMAS [8], and SNAP [9]. NUMAS creates a transition rate matrix for the model and constructs global balance equations from this. The balance equations are solved using a technique such as Gaussian elimination.

The Performance Analysts Workbench System (PAWS) is a commercially available package that uses simulation exclusively for the solution of EQN models. PAWS [10], [11] has several unique features. These include objects of type USER, INTERRUPT and SET. The USER object permits the user to enhance the modeling capabilities in essence by permitting arbitrary user specified actions when jobs visit that object. The SET objects allows dynamic service rates at a
server. A recent paper [12] describes a graphical user interface for PAWS. The implementation of this interface is not yet complete.

4. Thesis Goals

EQN models are characterized by a set of objects and their interconnections. Hence a natural way to specify EQN models is graphical. A graphical interface would ease the task of model specification and would reduce errors during the specification process. However, to our knowledge only in three cases have graphics interfaces for EQN modeling tools been considered. In the case of RESQUE, initial experience with a prototype implemented on an IBM personal computer was not satisfactory because of the large models often constructed by RESQUE users [13]. NUMAS designers reportedly intended to have such an interface by summer 1984. PAWS user interface is as yet unimplemented.

The Graphical Input Simulation Tool (GIST) is an attempt at providing a simulation-based EQN modeling tool with a graphical interface. Our aim in designing and implementing GIST has been to prove the feasibility of providing the user with a simple, intuitive, easy-to-use, easy-to-learn, and at least partially self documenting tool for solving EQN models. Such a tool also shields the user from details stemming from the design or implementation decisions made in building the tool.
Summary

Extended Queueing Networks permit modeling of a wide class of systems. Many tools have been developed for specifying solving EQN models. These tools differ in the manner in which they solve the models and also in the manner in which they accept specifications from the user. Though a natural way to specify EQN models is at least partially graphical most existing tools do not have such an interface. GIST is an attempt at proving the feasibility of providing the user with a simple, intuitive, easy-to-learn, easy-to-use tool for solving EQN models.
Overview of GIST

1. Introduction

The Graphical Input Simulation Tool (GIST) is a high level tool for EQN modeling that allows specification of models in a natural graphical manner. In this chapter we provide an overview of GIST. The various components of GIST and their interactions are discussed briefly.

GIST allows specifications of EQN models through a graphical or a textual interface and produces executable images to simulate a model of the system under consideration. The EQN models are specified as a network of high level objects drawn from a set of object types. GIST is based on CSIM, a subroutine package that provides runtime support environment for discrete event simulation. The major subparts of GIST are, a user interface that accepts specifications from the user, a translator that generates CSIM source code, and a library of object routines written in CSIM to model the various object types (figure 3.1). The user interface maybe graphic-oriented or menu-driven.

The following three sections discuss the components of GIST in detail. Section 2 discusses CSIM, the extended-C runtime support environment for discrete event simulation, on which GIST is based. Section 3 describes the abstractions supported by GIST as primitive objects. Section 4 contains a description of the user interface and
Figure 3.1  Organization and components of GIST
section 5 discusses the translator. A summary is contained in section 6.

2. Discrete Event Simulation and CSIM

EQN models can be solved analytically, numerically or by simulation. GIST uses simulation exclusively for the solution of EQN models. Specifications accepted through the user interface are ultimately converted to code for CSIM, a language-based simulation tool. Thus GIST can be thought of as a front end for CSIM. In this section we discuss discrete event simulation in general and then provide a brief overview of CSIM. For a complete description of CSIM the reader is referred to [3] and [14].

There are primarily two classes of simulation, discrete and continuous. Continuous simulation is applicable for systems in which the state of the system at a small fixed increment of time in the future can be extrapolated from the current state by using a suitable approximation of the differential equations governing the system.

Discrete event simulation, on the other hand, is more suited for systems where state changes occur at discrete points in time. Instances of such systems include queueing networks, where arrivals and departures of a job can be thought of as state modifying events. A common approach to implementing a discrete event simulator is to define a set of events that model state modifying actions, and a mechanism for scheduling these events. By definition no activity of interest occurs between events and this time interval is effectively
skipped over. This approach is known as the event scheduling approach to discrete event simulation.

A powerful enhancement to the event scheduling approach is known as process interaction. A process refers to a unique instance or activation of some user defined procedure. Processes, unlike events, have finite simulation time duration. Related sets of events can be combined into a single process increasing the conceptual ease of modeling the action of a system's components.

CSIM is a subroutine package written in the language C, that provides a runtime support environment for discrete event simulation and is based on the process interaction approach. CSIM provides special constructs such as Semaphores, State Variables, and Conditions and routines to manipulate these constructs. To write the simulation specification for a model, the user writes a set of procedures, one for each process and declares instances of special constructs. A preprocessor then scans through these special constructs to create representative data structures. Thus, CSIM can be thought of as a simulation compiler.

3. Object Routines

The implementation of the models produced by GIST consists of a set of procedures written in CSIM that simulate each of the primitive abstractions provided. There are two basic approaches to simulate a job flow system in CSIM. The components of the system can be modeled as processes and jobs can be modeled as tokens exchanged between
these processes. Alternatively, the components can be modeled as procedures that are executed by job processes, as the jobs flow through the system. The former approach has the advantage of being conceptually easier whereas the later results in fewer process suspensions i.e., fewer context switches. Suspending a process and activating another, though implemented efficiently in CSIM, is still a source of considerable overhead. Jobs in GIST are modeled as processes and most objects are modeled as procedures that are executed by these job processes, the exceptions being the SOURCE type object and the ALLOC type object.

The CSIM code that is produced by GIST basically consists of the following: declarations of special constructs of CSIM such as Semaphores, data structures for objects, initialization code, a job_process procedure, and a main procedure. The initialization code is meant for initializing any simulator data structures and special constructs. The job_process procedure contains calls to the various object routines specified for this model and the main procedure is the entry point for the executable image that ultimately gets produced. In essence, GIST produces CSIM code to simulate the specified model by putting together predefined object routines and some "glue" code. All object routines are precompiled to avoid the overhead of recompiling them for every new model.
4. User Interface

A unique feature of GIST is the approach taken in its user interface. Users can specify an EQN model either through the graphical interface or through a textual interface. Conceptually both interfaces provide the same capabilities i.e., the ability to specify the topology of a model, and the ability to define and edit parameters associated with each object. The following subsections give a brief outline of both the interfaces. The graphical interface is discussed in more detail in chapter 5.

4.1. Graphical Interface

EQN models are characterized by a network of objects and their interconnections. Thus a natural way to specify the objects relationship to one another is graphical. The Graphical User Interface and Dialogue Editor (GUIDE) allows users to specify EQNs by choosing from a set of icons representing object types. Users can define or edit object specific parameters conveniently using a series of "dialogue" windows.

GUIDE runs on a 512k Macintosh and resembles a typical application program on the Macintosh. The window environment, mouse input, and pull down menus of the Macintosh are employed in providing a friendly, easy-to-learn, easy-to-use interface. It runs standalone and no host computer is needed during the specification process. This reduces the load on the mainframe that runs the final simulation.
The output of GUIDE is a file containing the EQN model specifications. The file is transferred to the host for processing by the translator.

4.2. Textual Interface

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 meant to be used with a VT100 or equivalent terminal. It is based on the CURSIS windowing library.

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 such as RESQUE, interaction is through menus that appear in overlapping windows. This gives the user an improved sense of perspective about the current context in which he is during the specification process. The output of TIDE is again a specifications file to be processed by the translator.

5. Translator

The translator is the link between the user interface and the object routines. Its input is the specifications file output by the user interface and its output is CSIM source code. The translator serves to separate the model specification and model solution parts of GIST.
The translator produces as output the following files: a main file, a definitions file, a make file and optionally a report file. The main file contains code for initialization and calls to the appropriate object routines. The definitions file contains declarations of special CSIM constructs such as Semaphores and State Variables. The make file is provided to simplify the task of creating the final executable image and the report file contains user readable summary of the EQN model. A parser generated using the Unix facilities lex and yacc is used for parsing the user specified-conditions. The conditions that are parsed appear as Condition constructs of CSIM in the final code.

6. Summary

GIST serves as a front end to the simulation compiler CSIM that provides a runtime support environment and special constructs for discrete event simulation. The GIST software package consists of two user interfaces, a translator, and a library of object routines to simulate the various object types. The specifications entered via the user interface are converted to CSIM code by the translator and linked with precompiled object routines to produce the final executable image for simulating the system of interest. Jobs in GIST are modeled as processes of CSIM and most of the objects are modeled as procedures executed by the job processes.
Modeling Capabilities of GIST

1. Introduction

In analyzing the performance of computer and digital communication systems, the dominant factor is often seen to be contention for various resources such as processors, primary memory, secondary storage, and bus or communications link access. Queueing Networks (QNs) have been widely used to analyze and predict the performance of such systems. QNs, however, cannot be used to model systems where simultaneous possession of more than one resource by a job is permitted. Generalized or Extended Queueing Networks permit the modeling of such systems.

The elements of EQNs include a population of jobs, passive resources, and a collection of objects to model the various functions of a system. These objects can be classified according to the type of activities they can model: 1) active resources, 2) passive resources, 3) open systems, and 4) synchronization and concurrency.

GIST provides objects to model all of these activities. GIST differs from other performance evaluation tools in terms of modeling capabilities primarily in two ways. The first and most important distinction is that GIST provides two distinct objects for modeling the waiting and service functions required in dealing with active resources. This results in considerable flexibility in modeling complex systems. The second distinguishing feature of GIST is the avai-
lability of structured constructs for modeling concurrency and syn-
chronization. Separate objects for complex job routing and statis-
tics collection are also available in GIST.

This chapter discusses the modeling capabilities of GIST. Sec-
tion 2 discusses the various elements of EQNs such as jobs, and
resources. Some GIST-specific features such as conditions and
statistics collection are also discussed in this section. This is
followed by a discussion of objects for modeling active resources in
section 3. Section 4 contains a description of objects meant for
open systems modeling. Objects for modeling management of passive
resources are the subject of section 5, and section 6 discusses
objects for modeling synchronization and concurrency. Section 7 con-
tains a description of the objects meant for routing and statistics
collection.

2. **EQN Elements**

2.1. **Jobs and job classes**

One of the components of an EQN model is a population of jobs or
tasks. Depending on the nature of the real system being modeled,
jobs can represent various entities such as workpieces on an assembly
line, customers at a bank, or jobs in a computer system. Jobs posses
several attributes, the most important of which is the category or
class to which they belong. Job classes are important, since the
actions performed at certain objects when a job visits them, can be
job class dependent.
Another important attribute of a job is its relation to other jobs. Jobs can be independent or related to other jobs and a related job can be a peer or child/parent of another job. These attributes become important when synchronization and concurrency have to be modeled and are discussed in more detail in section 6. Jobs may possess more than one passive resources simultaneously.

Jobs do not change classes in GIST. There is no practical limitation in GIST on the number of jobs that can exist at any time. The number of jobs may be fixed or variable depending on whether the EQN model is open or closed. There is also no limitation on the number of passive resources a job may simultaneously possess.

2.2. Active and Passive resources

Contention for resources is often seen to be the dominant factor effecting the performance of certain types of systems. Such systems can be modeling using EQNs. Resources in such models can be represented by servers. Typically, a server can provide service to a single job at a time. Such resources, which are actively engaged in providing service, are known as active resources.

One of the limitations of the active resource abstraction is that, it can be used to model resources that serve only one job at a time. Often the need arises to model activities involving resources which are required by jobs before the jobs can receive service from an active resource. Such resources, which do not actively provide service, are known as passive resources. An example of a passive
resource is the primary memory in a computer system, some of which is required by jobs, before they can receive service from the cpu, an active resource. Passive resources can be conservative or non-conservative. If the amount of resource remains constant, the resource is known as a conservative resource, otherwise it is known as a non-conservative resource.

There is no limit on the number of passive resources that can exist in GIST and the number of passive resources a job can hold simultaneously at any time. Passive resources are allocated to jobs in integral quantities.

2.3. Statistics Collection

Good statistics collection mechanisms are essential in any simulation-based performance evaluation tool. Typical performance measures of interest in an EQN model include queue lengths, waiting times in queues and utilization of resources. Some measures, such as queue length, are associated with a particular object while others, such as inter-arrival time (the time between arrival of jobs at a particular object in the network) are more appropriately associated with the flow of jobs along a branch of the network.

GIST provides mechanisms to collect both types of statistics. Objects have the capability to collect specific object-related statistics. A special feature of GIST is the availability of an object type, PROBE, meant exclusively for statistics collection. PROBEs collect statistics on job flow along a branch. Measures such
as life time (time from creation of a job) and turn-around time (time between successive visits to a probe object) which can be thought of as characteristic of the network as a whole can also be collected by a PROBE object.

Since statistics collection entails considerable overhead, only the statistics requested by the user are collected. At present the mean and variance of a performance measure can be collected. This will be extended in the future to include histograms on the distributions.

2.4. Conditions

The ability to carry out actions based on the state of the model is essential in modeling complex systems. GIST provides this feature via the Condition mechanism. Conditions are logical expressions involving state variables. Objects with actions determined by Conditions are the SOURCE object, which can conditionally create a new job, and the SWITCH object, which can conditionally route jobs to different destinations.

All statistics collected are declared as state variables in GIST and are available for use in Conditions. This enables modeling of adaptive systems. For instance, in a multiprocessor system with load balancing, a new job can be routed to the processor with the least utilization or the shortest queue. Other variables that can be included in Conditions are Semaphores (in CSIM, a reference to a Semaphore is an implicit reference to the length of the queue of
processes waiting on that Semaphore).

2.5. Routing

Most job routing capabilities in GIST are concentrated in a single object type, the SWITCH. Except for the SWITCH object and the FORK and QUEUE type objects which have an implied routing of jobs on their output, every other object type has a single output i.e. jobs leaving that object are always routed to a single next object. This reduces the implementation effort of most of the objects considerably.

Routing between a QUEUE object and its associated SERVERs is complicated and is discussed in the next section. A FORK object routes incoming jobs along one branch and newly created jobs along another automatically.

The type of routing decisions possible at a SWITCH are probability-based, jobclass-based, or Condition-based. These are discussed in detail in section 7.

2.6. GIST object types

The following is a list of objects types GIST provides for modeling EQNs.

QUEUE: The QUEUE object models the waiting function (without service). Jobs arriving at a QUEUE wait until service is available.
SERVER: The SERVER object models the service function.

QSERVER: This object is composed of a queue and a server and is intended as an efficient alternative when the separation of service and waiting functions is not necessary.

SOURCE: Jobs of a single job class are generated at a SOURCE.

SINK: Jobs arriving at a SINK are removed from the network.

ALLOC: The ALLOC object models the acquisition of passive resources by jobs. Jobs requesting an allocation are delayed until the requested quantity is available.

DEALLOC: Jobs visiting a DEALLOC release the passive resource of a particular type if they possess any.

CREATE: A job's visit to a CREATE object causes the creation of a quantity of a particular type of passive resource.

DESTROY: Any passive resource of a particular type held by jobs when they visit a DESTROY is deallocated and is removed from the network.

FORK: Jobs of a particular class visiting a FORK create a new job. The new job may be a peer or a child of the creating job. Child jobs and their creators (parents) synchronize at a JOIN object.

JOIN: Jobs which have a parent/child relationship synchronize at this object. The child job is terminated and the parent job proceeds past the JOIN upon arrival of both jobs.
SWITCH: The SWITCH object routes jobs to one of several destinations using one of the following policies: probability-based, job class-based, and condition-based.

PROBE: The PROBE object collects statistics associated with the flow of jobs along a branch of the network. The icons for these object types are shown in Figure 4.1.

3. Active Resource Modeling

Contention for services is modeled in EQNs using queues. In other performance evaluation tools a queue is implicitly associated with a server; i.e., the waiting and service functions are modeled using a single abstraction. GIST departs from this approach and provides two separate objects, QUEUE and SERVER, to model the management of active resources. The QUEUE object models lines in which jobs wait to receive service and the SERVER object models the service function provided by an active resource. In addition to the QUEUE and SERVER objects, GIST provides a QSERVER object which is similar to the queue object in other tools.

GIST permits a single QUEUE to model contention for several services and conversely permits the modeling of contention for the same service at several QUEUES. Thus, it is possible to have a complex of QUEUES and SERVERs where every QUEUE routes jobs to every SERVER and every SERVER services jobs from every QUEUE (figure 4.2). Such modeling capabilities are not explicitly available in any other EQN.
### Figure 4.1 Object type Icons

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Q</td>
<td>SERVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CREATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SWITCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOURCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SINK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEALLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESTROY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JOIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROBE</td>
</tr>
</tbody>
</table>
performance evaluation tool to the best of our knowledge.

3.1. QUEUE object

A QUEUE object is characterized by two attributes. These are 1) queueing discipline and 2) Server selection rule.

3.1.1. Queueing Discipline

The queueing discipline determines the order in which waiting jobs become eligible for service. The rules for determining the eligibility for service are usually based on the time order of arrival of jobs, or job classes, or sometimes both. FCFS (first come first served) and LCFS (last come first served) are examples of queueing disciplines based on time order of job arrivals. Jobs can be grouped into priority classes based on their classes. Priority schemes can be preemptive or non-preemptive. A preemptive scheme is one in which jobs of a higher priority class are permitted to displace or preempt jobs of a lower priority class at a SERVER.

The following queueing disciplines are available at a QUEUE object:

1) FCFS (first come first served)

Jobs within each priority class leave the QUEUE in the order of their arrival, with jobs in a higher priority class departing before those in a lower one.

2) LCFS (last come first served)

A job that has most recently joined the QUEUE is the next one to
Figure 4.2 Complex QUEUE, SERVER Routing
depart. As in the previous case, jobs of a higher priority class depart before those of a lower priority class.

3) LCFSPR (last come first served with preemptive resume)
This discipline is the same as the LCFS discipline, with the difference that a job can interrupt the service of any previous job from the same priority class at any server. This preemption which is order based, is different from any possible priority based preemption that may exist at the QUEUE.

4) PS (Processor Sharing)
Only one priority class for all job classes may be defined for this discipline. All jobs share the service available at whatever server to which they are routed.

3.1.2. Server Selection
A SERVER is said to be available to a QUEUE if the QUEUE can route jobs directly to that SERVER for service. Since each QUEUE may route jobs to more than one SERVER, there arises a need for a policy to assign SERVERS to jobs. Server selection rules govern the assignment of jobs to available servers. The server selection can be either probability-based or priority-based. In either case all jobs at a QUEUE follow the same rule.

1) Probability-based Server Selection
A probability is associated with each available server for each job class. A new job entering a queue randomly selects one of the output
servers for this queue based on these probabilities.

2) Priority-based Server Selection
For each job class, the queue has an ordered preference list of servers for the purpose of server selection.

The statistics that can be collected at a QUEUE object are queue length and waiting time in the queue. In the current implementation, these are collected for all job classes as a whole. Extensions in the future will permit statistics collection on a job class basis.

3.2. SERVER object

The Server object in GIST is characterized by 1) Service time distribution functions, and 2) Queue selection rule.

3.2.1. Service time distribution function

A service time distribution (STD) function is the probability distribution from which service time values are drawn. STDs can be different for different job classes, but in the current implementation only a single STD can be defined for all job classes.

3.2.2. Queue Selection Rule

A Queue selection rule is the policy which determines the QUEUE object containing the job to be selected for service when a SERVER becomes idle. This is analogous to the server selection rule at a QUEUE object. Together these two policies determine the routing of jobs from QUEUES to SERVERs. The queue selection rule can be
probability-based or priority-based.

1) Probability-based Queue Selection
Each input queue to a SERVER has an associated probability coefficient. A SERVER, upon becoming idle, randomly selects a QUEUE based on these probability coefficients, from which it obtains its next job.

2) Priority-based Queue selection
Priority-based queue selection is similar to priority-based server selection at a Queue. Each SERVER has an ordered list of preferential input QUEUES and an idle SERVER uses this list to choose a QUEUE. If both the QUEUES and SERVERs use priority-based schemes for selection purposes, a stable-pair resolution is performed to resolve conflicts.

Statistics on a SERVER's utilization can be collected at a SERVER object.

3.3. Restrictions on Queueing Disciplines and Server Selection Rules

Certain restrictions are placed on the permitted combinations of selection rules at SERVERs, and on queueing disciplines at QUEUES in order that decisions to assign SERVERs to QUEUES can be made unambiguously. These rules are as follows.

[a] If the selection rule at a SERVER is probability-based, then either its input QUEUES all have a queueing discipline characterized
by sharing of available service (as in PS) or they all have nonsharing (as in FCFS or LCFS), nonpreemptive disciplines.

[b] If the selection rule at a SERVER is priority-based, then either (i) there is only one input QUEUE at each priority level, or (ii) in each priority level with more than one QUEUE, all QUEUES with that priority level have nonpreemptive disciplines, and either the QUEUES all have nonsharing disciplines or they all have sharing disciplines.

3.4. QSERVER object

The QUEUE and SERVER objects of GIST offer considerable flexibility in modeling complex systems. This flexibility, however, is achieved at the price of a fairly high computational overhead in simulation. The QSERVER object is provided as an alternative, when the extra flexibility of the separate QUEUE and SERVER objects is not needed. The QSERVER object models a queue with an associated server and is thus similar to queue objects in other performance evaluation tools.

The QSERVER, as its name implies is a single queue, single server complex. Incoming jobs join a queue, wait for the server to be free, receive service and depart. At present the only queueing discipline implemented is FCFS. No job class distinction is made at a QSERVER. The duration of service received by a job is a random number drawn from a probability distribution. Currently exponential and uniform job service time probability distributions are supported.
A variety of statistics can be collected at a QSERVER. These are 1) queue length, 2) waiting time in queue, 3) waiting time including service time, and 4) server utilization.

The QSERVER object is implemented very efficiently using the special CSIM constructs and should be used instead of the QUEUE and SERVER objects whenever possible. The rationale for implementing a separate QSERVER object is that a single queue associated with a single server is often encountered in EQN models. In the future, the QSERVER object will be extended to include other queueing disciplines and service distributions.

4. Open systems modeling

Queueing Networks in which jobs arrive from an external source, spend time in the network, and depart, are open. A closed QN, on the other hand, is one in which jobs remain within the system and move from object to object and the total number of jobs remains constant. An example of an open system is a batch-oriented computer system where a jobs arrive from the external world, are serviced by the cpu and associated I/O devices, and depart.

Modeling open systems requires the capability to model the arrival of jobs from an external source and the capability to model removal of jobs from the network. GIST provides two object types, SOURCE and SINK for these purposes.
4.1. **SOURCE object**

The **SOURCE** object models the arrival of jobs into the network from the external world. The **SOURCE** object operates in one of two basic modes: probability-based generation of jobs and condition-based generation of jobs.

In probability-based generation, the time between successive job generations, (the inter-generation time), is a random number drawn from a probability distribution. At present exponential and uniform distributions are available. A source with exponentially distributed inter-generation times, models a Poisson process, which is often encountered in queueing network models.

In condition-based generation of jobs, the **SOURCE** generates a new job whenever a user specified Condition is true. By constructing Conditions using statistics being collected elsewhere in the network it is possible to model adaptive arrival processes.

A **SOURCE** is allowed to generate jobs of a single class only. A source object can collect the following types of statistics: 1) inter-generation times, and 2) number of jobs generated.

4.2. **Sink object**

A **SINK** object models departure of jobs from a network. Jobs arriving at a **SINK** disappear from the network.

In the present implementation, it is an error for a job holding passive resources to visit a **SINK**. It is also an error if dependent
jobs (created at a fork object) visit a SINK.

No statistics collection capabilities are provided at a SINK. A SINK is a pseudo-object in the sense that unlike other object types no object procedure is associated with it. In the actual implementation there is only one SINK in the network. However, the user can specify any number of SINK objects in a model.

Certain bookkeeping functions are done at a SINK. These include releasing the space occupied by the job data structure and returning the process ids associated with the job processes for future use. These functions are transparent to the user.

5. Passive resource modeling

GIST provides four object types, namely ALLOC, DEALLOC, CREATE, and DESTROY to model management of passive resources. ALLOC and DEALLOC model acquisition and release of passive resources by jobs, and CREATE and DESTROY objects are intended for modeling non-conservative passive resources.

5.1. ALLOC object

The ALLOC object models allocation of a passive resource to jobs. Each ALLOC is associated with a single passive resource which it can allocate to visiting jobs.

The action at an ALLOC is job class dependent. Incoming jobs join one of several queues, each of which is associated with a prior-
ity class. In the current implementation there can be only be jobs of one class per priority queue. In the future this will be extended to generalized priority classes where jobs of different job classes can belong to a single priority class. Jobs that do not request any allocation pass through an ALLOC object without being delayed.

Allocation of resources is based on one of two policies: FCFS and First Fit.

1) FCFS: Jobs receive allocations on a first come first served basis in each priority class with an explicit priority ordering among priority classes. Thus no job in a lower priority class may receive units of the resource, until all higher priority queues are empty, and all jobs in its own priority queue that arrived earlier have received their allocations.

2) First Fit: Jobs are served on an FCFS basis as long as their requests can be satisfied. If any job’s request exceeds the available resource it is skipped over and the next eligible job is examined.

The amount of resource allocated to each job is an integer drawn from a probability distribution. The available distributions are exponential and uniform. Different distributions can be specified for different priority classes.

An ALLOC object can collect statistics on waiting time in the queue, and on the amount of resource allocated. Statistics can be collected for all job classes as a whole or for each priority class.
2. DEALLOC object

The DEALLOC object models release of passive resources held by a job. Each DEALLOC is associated with a passive resource and jobs that do not possess this resource pass through unhindered.

The function of a DEALLOC is job class independent. Whenever a job possessing the resource associated with a DEALLOC visits the DEALLOC, the resource is reclaimed from the job and is returned to a common pool for possible future allocation to other jobs. A job is never delayed by a visit to a DEALLOC.

Statistics on the amount of resource deallocated can be collected at a DEALLOC object.

3. CREATE object

A CREATE object models the creation of non-conservative passive resources. Each CREATE is associated with a passive resource. The action at a CREATE is jobclass-independent and whenever any job visits a CREATE, a certain amount of the resource is created and added to the common pool. The amount created is a random integer drawn from a user-specified probability distribution. The currently available distributions are exponential and uniform. Statistics on the amount of resource created can be collected at a CREATE.
§4. Destroy object

The DESTROY object is very similar to a DEALLOC object, the only difference being that the resource being deallocated from a visiting job is not returned to the common pool.

A job is not delayed by a visit to a DESTROY object. Statistics on the amount destroyed can be collected at a DESTROY object.

§5. Modeling Concurrency and Synchronization

GIST provides two objects, FORK and JOIN, to help model synchronization and concurrency. The FORK object can be used to model the simultaneous activation of related or unrelated jobs and the JOIN object can be used to model synchronization between related jobs. GIST, unlike other performance evaluation tools, imposes a structure on the usage of these objects for process synchronization.

The FORK and JOIN objects are the subject of the next two subsections. Subsection 6.3 discusses the structured usage of these objects.

§6.1. FORK object

Each FORK object has an associated job class. Whenever a job of that class visits the FORK, a new job is created. Jobs belonging to other job classes are unaffected by a FORK. A job is not delayed by its visit to a FORK.
The new job created at a FORK does not necessarily inherit any attributes of the creating job. No resources are preallocated for the new job. In addition, there is no restriction on the class the new job can belong to.

FORK is one of the few object types with an implied routing. Each FORK has two output paths and incoming jobs are routed along one of them and new jobs are routed along the other.

The relation of a newly created job to the creating job can be either peer or child. A peer job is independent of the creating job. A child job, on the other hand, maintains a relationship to its creator or parent job. Synchronization takes place between parent and child jobs at a JOIN object.

A parent job can visit several FORK objects and thus have more than one child job. A child job can also visit other FORKs, resulting in a hierarchy of jobs.

Peer jobs are appropriate for modeling systems in which events trigger other events without the need for synchronization at a later stage. An example is the transmission of an acknowledgement packet, upon receipt of a data packet, in a packet switching network.

6.2. JOIN object

A JOIN object models synchronization between related jobs in a system. Jobs that are independent of other jobs or do not have any related jobs (parents or children) proceed past a JOIN without delay.
When a job with relatives visits a JOIN, the following events take place. A parent job awaits the arrival of its recent non-terminated child job, if the child job is not already at the JOIN. When the child job arrives, the child job is terminated and the parent job proceeds to the next object. This is true even if the parent job itself has a parent, since synchronization between it and its parent will take place at the next join object.

No statistics collection capabilities are provided at a JOIN.

6.3. Usage of FORK and JOIN

GIST imposes a structure on the usage of FORK and JOIN objects for process synchronization. If a FORK is to generate peer jobs, no restrictions are imposed on it. However, for every FORK that generates a child job, there should exist a corresponding JOIN object. Thus FORKs and JOINs are constrained to occur in pairs when synchronization is to be modeled. Such FORK-JOIN pairs can be nested to any depth.

By nesting FORK-JOIN pairs in a structured fashion, it is possible to model any type of concurrency or synchronization that is possible with unstructured usage (figure 4.3). By unstructured usage, we mean usage in which there is no relation between the number of FORKs and JOINs. Structured usage results in a simpler conceptual picture being presented to the user and reduces the probability of incorrect usage. Moreover, most incorrect usages can be detected at either specification time or at execution time. In certain situa-
tions, there may be a larger number of objects required in the case of structured usage than in the case of unstructured usage. However, the advantages of conceptual ease of use and error detection capabilities of structured usage outweigh the disadvantages.

7. Routing and statistics collection

GIST includes two more object types that are usually not found in other EQN modeling tools. These are the SWITCH and the PROBE object types.

7.1. Switch object

The SWITCH object is intended for modeling complex job routing patterns. Each SWITCH is associated with one or more output paths and incoming jobs are routed along one of these paths without any delay.

A SWITCH routes jobs according to one of the following routing policies: 1) probability-based, 2) class-based, and 3) condition-based. In the current implementation mixing different policies in the same SWITCH is not permitted. This is not a limitation, since by using a cascade of SWITCHes, it is possible to model any mix of the allowed policies.

7.1.1. Probability-based Routing

An incoming job is randomly routed along one of the output paths using user specified probability coefficients associated with each
Figure 4.3  Nested Synchronization
path. No class distinction is made in this policy under the current implementation.

2.1.2. Class-based Routing

Each output path is associated with a single job class and incoming jobs are routed along the appropriate path based on their class. If an incoming job has a class that is not associated with any destination then it is routed to a default destination. The default destination is user-specified.

2.1.3. Condition-based Routing

A Condition is associated with each output path. An incoming job causes the evaluation of these conditions to occur in a predetermined order. The job is routed along the path associated with the first condition which evaluates to true. If all conditions are false, the job is routed to a default destination as in class-based routing.

Statistics on the number of jobs routed along any path can be collected at a SWITCH.

2.2. PROBE object

The PROBE object is solely intended for statistics collection. Statistics which are associated with the flow of jobs along a branch of the network, such as inter-arrival times, and statistics which are characteristic of the network as a whole, such as turn-around times,
can be collected by the PROBE. The PROBE, in effect meters the job flow along a branch. Jobs are not delayed by PROBEs.

The statistics collected at a PROBE are collected on a job class basis. At present, the following statistics can be collected at a PROBE:

1) Inter Arrival Times: The interval between successive arrivals of jobs belonging to a particular class.

2) Job Count: The number of jobs of that have visited this probe thus far.

3) Resource Held: The amount of passive resource of a particular type held by jobs.

4) Life Time: The elapsed time between a job's creation and its visit to the PROBE.

5) Turn Around Time: The time between successive visits of a job to this PROBE.

The PROBE object, like most other objects is implemented as a procedure invoked by a job process. Its utility derives in part from the fact that it is often conceptually easy for a user to envision a metering device along a branch of the network. Turn-around times are collected by tagging jobs with special data structures and result in considerable overhead. In the current implementation, a job can carry only one such tag, thus limiting the number of turn-around times statistics that can be simultaneously collected on a single
job.

8. Summary

GIST offers users a variety of useful features for modeling EQNs. These include, the ability to model open systems, an extensive set of objects for managing passive resources and the ability to model synchronization and concurrency. Some of these features are available in other EQN modeling tools. GIST, however, is unique in several respects.

The separation of service and waiting functions into two distinct objects, to our knowledge, has not been implemented in other performance evaluation tools. GIST also provides a structured facility for modeling synchronization and concurrency in the form of the FORK and JOIN objects. GIST permits modeling of complex routing mechanisms. The PROBE object allows collection of statistics that are relevant to more than one object.

One of the aims in designing and implementing GIST has been to provide a very high level modeling tool of reasonable efficiency. We feel that this objective has been accomplished by the object repertoire which GIST provides.
Graphical Interface

1. Introduction

As programs become increasingly sophisticated, the interaction between the user and the program becomes more complicated. The design of the user interface in many cases has not kept pace with the increased sophistication of the user/program interaction. However there has been considerable interest over the past few years in the area of sophisticated user interfaces. This is evidenced by the recent trend towards graphical user interfaces and away from purely textual interfaces. The availability of workstations such as the Sun workstation from Sun Micro Systems, and low cost personal computers such as the Macintosh personal computer from Apple computers, with window environments on high resolution displays is an indication of such a trend.

Some of the characteristics of a good user interface are: 1) it should be easy to learn 2) it should be easy to use 3) it should not intimidate the user 4) it should hide irrelevant details from the user 5) it should be consistent. By consistency we mean that similar actions on the part of the user should lead to similar results. These factors however, are not independent and interact with each other.

EQN models are characterized by a network of objects and their interconnections. Users almost always draw a pictorial representa-
tion of an EQN model before actually specifying it to a typical modeling tool. Such representations or block diagrams are much more readable than textual information accepted by EQN modeling tools. In effect, users manually translate a graphical representation into a suitable form. Such translation is time consuming and is a potential source of errors during the specification process. An obvious way to bypass this step is to provide the user with an interface that directly accepts graphical specifications. However, certain information associated with EQNs is necessarily specified textually. Examples of such information include the attributes of each object such as coefficients for probability distributions, names of objects and initial conditions.

The primary motivation in developing GIST has been to prove the feasibility of providing the user with a simple, intuitive, easy-to-learn and easy-to-use tool that allows specification of EQN models in a partially graphical manner. A good graphical user interface is essential in achieving these ends. Such an interface shields the user from implementation details and reduces the probability of errors during the specification process. It lets the user concentrate on the model specification rather than on the tool itself.

We discuss the graphical user interface of GIST in this chapter. Section 1 gives an overview of the interface. Section 2 discusses the interface in detail and provides the rationale for some of the decisions that went into its design. Section 3 illustrates the usage of the interface through an example and Section 4 contains a summary.
2. Overview of the Graphical Interface

The Graphical User Interface and Dialogue Editor (GUIDE) accepts specification of an EQN model by the user and produces a specifications file to be later processed by the translator component of GIST. It has been implemented on a 512 Kbyte Macintosh personal computer and at present occupies about 120 Kbytes. GUIDE runs standalone without the need for a host computer during the specification process.

GUIDE is written in the C language, using the Stanford University Mac C (sumacc) development software. It is tailored to the Macintosh and is not portable to other computers/graphics terminals.

Users familiar with some typical Macintosh application programs can learn to use GUIDE in a few minutes. It however, does not teach the user about EQN models. Provision has been made to give on-line help about the various objects of GIST, but a present this feature is unimplemented.

3. Design of the Interface

We discuss the design and operation of GUIDE in this section. We first give the rationale for our choice of the Macintosh for implementing GUIDE. Then we provide an overview of the software and a brief description of the operation of GUIDE concludes this section.
3.1. Hardware Requirements for Interface Implementation

The minimum hardware requirements for a good graphical interface are, a high resolution display, sufficient memory to specify large models, and a graphical input device such as a mouse, trackerball or a digital pad along with a keyboard. Several workstations and personal computers satisfy the above requirements. Examples are the Sun workstation from the Sun Micro Systems and the Macintosh family of computers from Apple computers. We chose the Macintosh computer primarily because it is the least expensive alternative which meets these minimum requirements, and is more widely available than more sophisticated (and more expensive) workstations. Moreover, the Macintosh has an excellent library of graphics routines and a well defined and widely accepted user interface, that has reduced our implementation effort considerably.

3.2. Implementation Overview

The software for GUIDE can be divided into two parts: a high level graphical part and a set of specification routines. The graphical part allows the user to create a graphical representation of the network 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 in windows tailored to each object type.
These routines have been implemented in a modular fashion, making it easy to modify existing object types or add new object types.

3.3. Interface Description

On invocation, GUIDE presents the user with a set of menus, a working window and an options window (figure 5.1). The working window is the area of the screen in which the user creates a representation of the model. This window can be scrolled horizontally and vertically to help in specifying large models. Thus models need not be limited to one screen size. The options window contains graphical representations (icons) for each object type available in GIST. The icons for the object types are given in figure 5.2.

There are seven menus: 1) the apple menu, 2) the file menu, 3) the edit menu, 4) the help menu, 5) the specify menu, 6) the transfer menu, and 7) the debug menu. Moving the cursor over any menu item and pressing the mouse button results in the display of a set of items grouped under this menu (pull down menus). Any item can then be invoked by moving the cursor over it and releasing the mouse button. The apple menu is the standard menu item found in other typical Macintosh applications and is not directly relevant to the interface, but has been provided to maintain consistency with other Macintosh applications.

The file menu lets a user invoke the following items: open a new file, open an existing file, close current file, save current specifications in a file, save current specifications in a different file, revert back to the last saved version of the specifications, check
Figure 5.1 GUIDE Windows and Menus
Figure 5.2 Object type Icons
Figure 5.3 Dragging an Icon into working window
After some or all of the required icons, (in this case), SOURCE, QSERVER, PROBE and SINK have been dragged into the working window we can specify the interconnections between them using the interconnect option in the options window. If the interconnection option is selected the cursor changes shape to a cross inside the working window to keep the user aware of this fact (figure 5.4). Interconnection is specified by clicking first on the source icon and then the destination icon. Source icon is the icon representing the object that is to route exiting jobs and destination icon is the icon representing the object that is to receive these jobs. Thus, in our case the source icon and the destination icon in figure 5.4 are the SOURCE object icon and the QSERVER object 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. If the source and destination icon are same, the interconnection is undone.

Object type specifications can now be given for each object in turn. Clicking twice in succession (double clicking) on an icon in the working window automatically brings up a "dialog" window specific to the object type being represented by that icon. These windows contain boxes to accept object names (text boxes), items that permit selection among a number of alternatives (rad controls), items that accept yes/no options (check items) and items that start an immedi-
Figure 5.4 Specification of Interconnections
ate action (buttons).

As example double clicking on the SOURCE object icon brings up the SOURCE specification dialog window in figure 5.5. This window contains text boxes for the name of the object, the class of jobs that can be created by this SOURCE, and optionally boxes for distribution parameters. The interface is designed to hide as much irrelevant detail as is possible from the user. Thus, choosing the conditional generation option hides the distribution related items in the window and brings up a text box to accept the condition. Other items in the dialog 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. This closes the dialog window and takes the user back to the working window. Alternatively, clicking on the cancel button undoes all changes made to this object’s specification and returns the user to the work window. This undo feature is available in all specification dialog windows.

Double clicking on the QSERVER object brings up another dialog window (figure 5.6). Similarly, the specifications for the PROBE can be given through a dialog window. The PROBE dialog window (figure 5.7) contains some additional button items “Add Class”, “Delete Class”, and “Next Class”. These can be used to specify the list of job classes for which this PROBE is to collect statistics. The SINK object does not require any specifications.
Figure 5.5 SOURCE specification dialog window
Figure 5.6 QSERVER specification dialog window
Figure 5.7: PROBE specification dialog window
Specifications about runtime parameters such as duration of simulation can now be given through another dialog window which is invoked by selecting the "Run Parameters" item in the specify menu. This brings up a Run Time parameters specification window with a text box to accept the simulation time (figure 5.8). 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.

The model specification is now complete. This file can be saved and then transmitted to the host computer for processing by the translator.

5. Summary

The Graphical Interface Dialogue Editor (GUIDE) for GIST runs standalone on a Macintosh personal computer without the need for a host computer during the specification process. Extensive use has been made of the Macintosh features such as overlapping windows, menus and mouse input, in providing the user with an intuitive, easy-to-learn and easy-to-use interface.

Users typically specify the topology of an EQN model using GUIDE by selecting icons representing the required object types and placing them in a work area. Interconnections are specified in a natural way by clicking in succession on source and destination icons. Specifications for an object are given via dialog windows that contain items to accept data specific to this object data.
Figure 5.8 Runtime parameters specification dialog window
GUIDE obviates the need for manual translation of graphical representations of EQN models by directly accepting specifications in a graphical fashion. This reduces the time required to specify the model and also reduces the probability of errors during the specification process.
Modeling with GIST

1. Introduction

We discuss some examples of GIST usage in this chapter. Through these examples we illustrate the following modeling capabilities of GIST: open systems modeling, management of passive resources, job routing, synchronization and concurrency, and separation of service and waiting functions.

Section 2 discusses a multiprogrammed computer system. This example is primarily intended to demonstrate modeling contention for passive resources using the GIST objectsALLOC and DEALLOC. Routing capabilities of GIST are also illustrated in this example. Section 3 contains an example of a distributed database system. Usage of FORK and JOIN objects to model nested synchronization is main the topic of this section. In section 4 we illustrate the utility of separating the queueing and service functions by considering a multiprocessor system.

These examples are provided to illustrate the ability to use GIST to model complex systems. However, this is intended neither to be a tutorial on GIST nor to be an exhaustive survey of the modeling features of GIST. In these examples, we omit detail not directly related to the purpose of the discussion.
2. Modeling contention for passive resources

The model presented in this section serves to emphasize three features of GIST. These are contention for passive resources, complex job routing, and open systems.

Consider a simplified multiprogrammed computer system with batch tasks and interactive tasks. Batch tasks enter the system, acquire primary memory, receive service from the processor, release the primary memory they have acquired, and depart. Interactive tasks acquire primary memory, receive service from the processor and return after undergoing further processing (possibly at a terminal). Statistics of interest in such a system include, response times for interactive tasks and throughput for batch tasks. Typically, interactive tasks in such systems share the processor so as to reduce the average response time.

This system can be modeled using the GIST objects ALLOC, DEAL-LOC, QUEUE, SERVER, SOURCE, SINK and SWITCH. Primary memory can be modeled as a passive resource which is contended for by jobs of two classes representing the two types of tasks in the real system. Jobs of class interactive represent interactive tasks and jobs of class batch represent batch tasks. The details of the model (figure 6.1) are given below:

The SOURCE object BS models the arrival of batch jobs into the system. BS uses a probability-based job generation rule and generates jobs of batch class. The SINK object is used to represent the departure of batch tasks.
**Figure 6.1** Multiprogrammed computer system

- **BS** Batch source
- **MA** Memory allocate
- **QB** Batch queue
- **Cpu** Cpu server
- **Sw1, Sw2** Switches
- **SINK** Batch sink
- **MD** Memory deallocate
- **QI** Interactive queue
- **P** Probe
- **IJP** Interactive job process
The ALLOC object MA models the contention for the passive resource memory. A priority-based First Fit allocation policy is employed with interactive jobs enjoying the higher priority.

The SWITCH object SW1 routes incoming jobs on a job class basis. Interactive jobs are routed to the QUEUE object QI and batch jobs are routed to the QUEUE object QB.

QI represents the interactive job queue at the processor and QB represents the batch job queue. The queueing discipline at QI is processor sharing whereas the discipline at QB is FCFS. Both QUEUE objects select the single SERVER object Cpu.

The SERVER object Cpu represents the processor. Queue selection rule at this object is priority-based with QI being at a higher priority.

Jobs exiting Cpu are routed to the PROBE object P which collects statistics on turn around times for interactive jobs, and statistics on throughput for batch jobs. Jobs are routed to the DEALLOC object MD after their visit to P. MD models the release of primary memory by tasks departing the processor.

The SWITCH object Sw2 routes jobs on a job class basis. Batch jobs are routed to the SINK while interactive jobs are routed back to the ALLOC object through the block UJP.

Apart from the activities of the tasks explicitly stated in our discussion, a more detailed model of the system would certainly refer to other activities undergone by an interactive task.
interest of simplifying the discussion we have represented these activities by the block IJP (interactive job processing) in our model.

3. Nested synchronization

In this section we will illustrate the usage of the GIST objects FORK and JOIN in modeling nested synchronization, through an example of a distributed database system.

One of the primary concerns in the design of database systems is the need to keep the state of the database consistent. A mechanism frequently used for this purpose is that of locks. Transactions that need to access a set of data items have to place locks on these items before any further processing. Locks prevent transactions from reading intermediate values created by other transactions. Also, in certain systems, locks that have been in existence for a long period of time may be broken when other transactions request locks. Typically, the original transactions that have placed the locks are notified about the violation of these locks under such circumstances.

Consider a distributed database system in which data items are scattered among several machines. Locking of data items on different machines may proceed in parallel in such a system. Since a transaction cannot proceed unless it has placed locks on all data items to be accessed, some form of synchronization is necessary among all the different processes that are placing these locks. This can be modeled using the GIST objects FORK and JOIN with jobs representing
transactions. Figure 6.2 represents a scheme that models these operations.

Jobs entering the FORK object F1 create child jobs which are routed to F2. The parent jobs are routed to P1. The parent job represents the process of acquiring locks of data items on the host machine, and the child job represents the process of acquiring locks of data items on a second machine.

Jobs entering the FORK object F2 create children which are routed to P3. The parent job which is routed to P2 represents the process of placing locks on data items on the second machine, whereas the child job represents the process of notifying other transactions of possible broken locks.

Since our primary aim is to illustrate nested synchronization, we have represented the actual activities that take place during locking or notification of broken locks as blocks P1, P2 and P3. Also, in the interest of simplicity, we have shown these blocks to be disjoint although no such restriction exists in GIST.

Jobs reaching the JOIN object J2 wait for related jobs. Upon arrival of both the parent and child jobs, the child job is terminated and the parent job proceeds to JOIN object J1. This represents the synchronization that takes place between the processes placing locks and notifying other transactions of broken locks, on the second machine.
Figure 6.2 Modeling nested synchronization

F1, F2 FORKS
J1, J2 JOINS

P1 Acquire locks on Machine A
P2 Acquire locks on Machine B
P3 Inform other transactions of broken locks
As in the case for J2, jobs reaching the JOIN object J1 wait for related jobs. Upon arrival of both the parent and child jobs, the child job is terminated and the parent job, which represents the original transaction, proceeds past the JOIN. Nested synchronization is thus achieved.

4. Separation of Waiting and Service functions

We now demonstrate the usage of the QUEUE and SERVER objects through several variations of an example of a multiprocessor system.

Consider a heterogeneous multiprocessor system with two processors, one of which has a higher performance than the other. As in the earlier example, this system serves two types of tasks: interactive and batch. Batch tasks are restricted to execution on the low performance processor whereas interactive tasks may execute on either. This system can be modeled using the GIST objects QUEUE, SERVER, SOURCE, SINK, and SWITCH. Two classes of jobs, interactive and batch, represent the two types of tasks in the system. The details of the model (figure 6.3.1) are given below.

This SOURCE object SB models the arrival of batch jobs into the system while the SINK object models their departure.

The SERVER object A1 represents the high performance processor, while the SERVER object B1 represents the low performance one. A1 receives input from the QUEUE object Q1 and the QUEUE object QB whereas B1 receives input only from Q1.
Figure 6.3.1 Multiprocessor system with 2 processors

Figure 6.3.2 System with one additional processor

Figure 6.3.3 System with two additional processors
The QUEUE object QI is for interactive jobs and the QUEUE object QB is for batch jobs. The queuing discipline at QI is processor sharing whereas that at QB is FCFS.

The SWITCH object SW routes batch jobs to the SINK and interactive jobs to the block UP which represents some additional activity interactive jobs undergo before they return for service.

The PROBE object collects statistics on turnaround times for interactive jobs and throughput for batch jobs.

If the performance of this system is deemed inadequate, it can be enhanced by adding processors. Typically, financial considerations limit the number and nature of processors that can be added. In such cases, performance analysis of each of the various possible configurations can help choose between alternatives. For the system modeled above, we consider how two alternatives in this regard may be modeled.

Figure 6.3.2 represents the model of the original system with an additional high performance processor A2. Interactive jobs can now execute on either A1 or A2 while batch jobs can execute on either A1 or B1.

Figure 6.3.3 represents an alternate configuration with two additional low performance processors B2 and B3. B2 is solely available for the execution of interactive jobs while B3 is available for the execution of batch jobs.
This example demonstrates the flexibility provided in modeling complex systems because of the separation of waiting and service functions in GIST. Specifications for one model can be transformed into that of another model with ease.

5. Summary

We have presented three examples of GIST usage dealing with the various modeling features of GIST. These features include, open systems, synchronization and concurrency, and complex job routing. Systems which exhibit complex patterns of contention for active resources can be modeled with ease in GIST.
Conclusion

GIST was designed with two principal objectives. These were to prove the feasibility of providing the user with an easy-to-learn, easy-to-use, high level specification tool that accepts specifications in a graphical manner, and to provide a high level, versatile tool that is complete with respect to EQNs. Although GIST is not fully implemented, and a number of extensions are planned or proposed for it, our experience with the development of the current implementation of GIST is sufficient to allow us to draw some conclusions about how effective GIST is in fulfilling these objectives.

We conclude the thesis with a status report and an evaluation of GIST. We discuss the strengths and weaknesses of GIST and our approach in building it and provide some suggestions for future work.

Section 1 contains a report on the current status of the GIST software package. A discussion of the relative merits and demerits of the transaction-oriented approach as it relates to the design of GIST, is contained in section 2. In Section 3 we evaluate GIST in terms of how successfully it has achieved our original goals. Section 4 discusses the areas in which GIST can be applied. Some suggestions for future work are included in section 5.
1. Status Report

All three major components of GIST, namely the object routines, the translator and the user interface (graphical and textual), are currently implemented and have been tested in isolation. Some minimal testing of the package as a whole has also been done. In the near future, we plan to carry out extensive testing of the entire package and release it to the user community at Rice. Feedback from the user community is expected to help result in modifications to the existing objects and addition of new object types. Our limited experience has uncovered some weaknesses due to restrictions placed by us on the objects and also due to the lack of certain object types. These will be discussed in more detail in section 5. We hope to correct some of these deficiencies in the near future before we release GIST to the user community.

2. An Evaluation of the Transaction-Oriented Approach

In a transaction-oriented language or tool, the user is presented with a limited set of abstractions. Typically, these are tailored for a special purpose. Two of the major advantages of tools based on this approach, as opposed to a general purpose tool, are the savings in effort and time for the user. The savings in effort results from the fact that often encountered situations or problems can be succinctly specified or solved using the available set of transactions. For instance, a Poisson source which is a common feature of open EQNs, can be directly modeled by a the SOURCE object
in GIST. The savings in time result because models can be specified quickly and in a relatively error-free manner, provided a reasonably good user interface exists for the tool.

These advantages, however, are not without attendant drawbacks. The most prominent of these drawbacks is that the tool is relatively inflexible. The user is constrained to use the set of abstractions provided by the implementor. Thus, a transaction-oriented tool, by design, will not be able to solve or model all systems that a general purpose tool will. Another drawback of a transaction-oriented tool is its relative inefficiency when compared with general purpose tools. Since the limited set of transactions have to be as general as possible, a user will usually be able to solve his problem more efficiently in a general purpose tool. A user may be able to produce more efficient code for input to CSIM than GIST does for the same model.

However, we believe that in the case of GIST, the advantages outweigh the disadvantages. As an analyst's time becomes more valuable and as processing time becomes less expensive, such tools appear even more attractive. Also, some of the limitations resulting from the transaction-oriented approach can be overcome. We present some suggestions for doing this in section 5.

3. An Evaluation of GIST

One of the primary goals in designing and implementing GIST was to prove the feasibility of providing a graphical input tool for
specifying EQN models. The other main objective was to provide a very high level tool that is complete with respect to EQN models. We believe that the current implementation of GIST conclusively proves the feasibility and utility of such a tool. In terms of modeling capabilities, GIST provides analogs of most object types commonly found in other high-level EQN modeling tools, and also includes object types that have no counterparts in other tools.

GIST differs from other EQN modeling tools in the manner in which users input specifications and in the capabilities it provides for modeling complex systems. One of the major strengths of GIST is its graphical user interface. GUIDE accepts specifications in a manner that is very natural and intuitive. Most syntactic and some semantic errors are made impossible, ensuring a relatively error-free specification process. TIDE is a convenient alternative in the case of the unavailability of a Macintosh. TIDE is almost as convenient to use as GUIDE for specifying object characteristics and runtime parameters. Thus a GUIDE-designed model can have its parameters varied easily using TIDE, when the goal is to evaluate the same model under different workloads and/or object capacities or capabilities. In the current implementation there are some minor differences in format that prevent editing a GUIDE-designed model using TIDE. This will be corrected in the near future.

In the area of modeling capabilities GIST differs from other tools primarily in two ways. The first is the unique feature of GIST in providing two separate objects to model the service and waiting
functions. Systems which do not fit into the traditional notion of a combined waiting and service center can be modeled with ease in GIST. The second feature allows modeling of nested synchronization by structured usage of FORK and JOIN objects. GIST also provides a set of four objects for managing conservative and non-conservative passive resources. Other notable features of GIST include the routing and statistics collection abstractions SWITCH and PROBE. GIST thus provides most of capabilities needed to model BQNs.

Some of the limitations of GIST result from the specific approach we have taken and are inherent to transaction-oriented tools. These deficiencies include relative inflexibility, inefficiency, and loss of generality. GIST is inflexible in the sense that a user is restricted to the available set of objects. Adding a new object or modifying an existing one is nontrivial and should be attempted only by a very sophisticated user or more likely not at all. Due to the automatic nature of the code generation, GIST will typically produce less efficient code than will a knowledgeable user of CSIM. Also, all systems that may be modeled in a language-based tool like CSIM cannot be modeled using GIST. This loss of generality is again an inherent limitation of transaction-oriented tools.

Apart from the limitations resulting from the approach used in designing GIST, other limitations, mainly due to lack of suitable object types or inadequate capabilities of the existing ones, are also present. These include the lack of job variables, the inability
of the DEALLOC and DESTROY objects to remove a partial quantity of the passive resource held by a job, the class-independent nature of a SERVER object, inability to communicate between jobs not in a parent/child relationship, and limited repertoire of distributions. Most of these deficiencies are only a reflection of the current implementation and can easily be corrected. Several improvements are suggested in section 5. Despite these limitations, GIST is a highly capable tool for modeling systems with EQN models.

4. Applications of GIST

GIST has applications in three main areas: research in the design and organization of computer systems and computer networks, teaching the principles of modeling and performance evaluation of such systems, and research in the design of better performance evaluation tools and machine architectures to support such tools.

Often in courses relating to modeling and performance evaluation students spend a considerable amount of time coding and debugging simulation models in either a general purpose language or a language-based simulation tool. This severely restricts the number and complexity of systems that can be analyzed over the duration of a course. A high-level tool like GIST can drastically reduce the time required to analyze a system.

Another use we envisage for GIST is that of a vehicle for testing new ideas in the area of performance evaluation tools. One of the simulation design alternatives being investigated at Rice in the
area of architectural support for distributed discrete event simulation is that of object oriented simulation. It is hoped that extracting parallelism out of simulation models specified using such tools would be considerably simpler than doing so in the case of a general purpose tool. One possible way to parallelize GIST would be to implement all GIST objects as processes and execute them concurrently on different processing elements. This, however, is a significant departure from the current implementation of GIST and requires a great deal of further investigation. One of the advantages of making the implementation of GIST completely transparent to the user is that changing the implementation of the underlying object routines from procedures to processes will not effect the user. Thus one can experiment with alternate implementations, even creating a version of GIST designed for producing simulation models to be executed on parallel processors.

5. Future Directions

GIST can be enhanced in several ways. New objects can be added or existing objects can be modified to enhance their capabilities. Other solution methodologies for EQN models can be incorporated into GIST. The user interface can be enhanced by adding on-line and sub-model specification features. Finally, the approach used in GIST can serve as a basis for GIST like tools in other areas.

In this section we discuss some possible enhancements to the capabilities of GIST and make some suggestions for overcoming some of
its limitations.

5.1. Additional capabilities

The modeling capabilities could be enhanced in several ways: by adding new object types, by modifying existing object types, and by providing job variables and global variables. Some of these features are listed below.

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. Permitting jobs to carry additional numerical information and basing the actions at some objects on this information allows modeling of systems where tasks retain a certain amount of history. Jobs could be made to carry some information in the form of job variables. Objects which could be job variable dependent include QSERVER and SERVER where the service time could be job variable dependent, ALLOC, DEALLOC, CREATE and DESTROY where the amount of passive resource manipulated could be job variable dependent, and SWITCH, which could have a job variable based routing policy. Job variables can be manipulated by a new object type ASSIGN.

In addition to retaining some history of each job in the form of job variables, a certain amount of 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 considerable flexibility in modeling. Global variables
could be made available for use in condition construction.

New object types that can be added include ASSIGN, WAIT, and DELAY. The ASSIGN object modifies the values of global variables or job variables of visiting jobs. The WAIT object makes jobs wait on the values of conditions. A DELAY object delays a job for an amount of time drawn from a probability distribution.

Some of the modifications that could be made to the existing object types are listed below.

Additional queueing disciplines at a QSERVER.

Class based statistics collection at QUEUE and SERVER.

Class based service time distributions at a SERVER.

Generalized priority queues and additional allocation policies at an ALLOC.

A variable amount of resource to be deallocated or destroyed at a DEALLOC or DESTROY object.

Creation of multiple peer or child jobs by a single job at a FORK.

Job variable based routing at a SWITCH.

Analysis and validation of simulation results, such as confidence interval estimation is invaluable to the user. At present, no such provision exists in GIST. Addition of this capability should be given a high priority in any future projects involving GIST.
An important enhancement that can be made is the provision of submodels. A submodel can be thought of as a new object type composed of existing object types. For instance, a submodel composed of QUEUEs and SERVERs to model a uniprocessor could be used in building the model of a multiprocessor system. Submodels help in encapsulating and hiding detail and let the user reutilize some earlier work. This makes it possible to have libraries of submodels tailored to specific areas such as computer networks.

One of the drawbacks of the transaction-oriented approach is that the user is limited to the available set of abstractions. A way to overcome this limitation in GIST is to provide a new object type USER, that can in essence be specified by the user. Jobs visiting this object invoke some user specified procedure or process. This may require that the user write some amount of code in CSIM, but the flexibility afforded may outweigh the loss of convenience.

5.2. Enhancements to the user interface

Enhancements possible in the area of user interface include on-line help feature, submodel specification capability and display of simulation results.

On-line help can be provided to the user in a number of areas such as assistance in using the interface itself and assistance in using GIST objects. It may also be possible to provide a guide to the use of models in general. Provisions have been made to add on-line help to the graphical interface, and this feature will be imple-
mented in the near future.

At present the results of the simulation are output on the host machine and the user interface is not involved in the display of these results. It would be convenient for the user to deal with a single entity, the user interface, for all interaction with GIST.

Another possible extension is the generation of a report of the simulation results along with a brief description of the model, i.e. a concise report of the model.

5.3. Alternative methods of solution

Simulation is but one means of solving EQNs. Alternate numerical methods of solution such as Mean Value Analysis (MVA) would be extremely helpful to the user. The existing translator could be modified to check for applicability of the various methods. Thus, one can envision a general purpose translator, and a set of solvers one of which would be the current code generator for CSIM.

5.4. Application to other tools

The transaction oriented approach is appropriate in a number of other areas. Modeling applications where complex entities can be constructed out of a limited set of abstractions, such as logic design, Petri Nets, and process control, would benefit from a transaction oriented tool. Thus it is entirely possible to have GIST like tools for solving Petri Net models or gate level logic designs. One such tool, for gate level verification of logic designs, already exists at Rice. At a higher level one could think of a tool tailored
to the analysis of computer architectures. At the extreme would be a hybrid tool that permits specification of models using a mix of the above mentioned methodologies.

6. Conclusion

Despite some limitations, GIST is a versatile and convenient tool for E\(N\)N modeling. Some of the limitations of GIST are inherent to the transaction oriented approach taken in designing it. Unique features of GIST include its dual user interfaces and its special modeling capabilities such as separation of service and waiting functions and nested synchronization. We believe that we have achieved the original goals we have set ourselves in designing GIST. The feasibility of providing a high level graphical specification tool for E\(N\)Ns has been established. We now have a tool at our disposal that can be used to further research in the areas of performance evaluation tools, and performance evaluation in general.
Bibliography


