TOWARDS A GRAPHICAL QUEUEING NETWORK TOOL

By

JAGANATH DABBI

Bachelor of Technology (Honors)
Indian Institute of Technology
Kharagpur, India
1985

Master of Business Administration
Oklahoma State University
Stillwater, Oklahoma
1993

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirement for
the Degree of
MASTER OF SCIENCE
May 1994

OKLAHOMA STATE UNIVERSITY

TOWARDS A GRAPHICAL QUEUEING NETWORK TOOL

Thesis Approved:

M. Samadzadeh-H.

Thesis Advisor

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my appreciation and gratitude to my advisor Dr. Mansur H. Samadzadeh for accepting to be my major advisor, his advice, intelligent guidance, and assistance. His constructive criticism, direction, wisdom, and counsel during my graduate study have been a constant source of inspiration and motivation that helped me gain confidence. I also wish to thank Drs. Blayne E. Mayfield and D. Paul Benjamin for serving on my graduate committee.

Additionally, I wish to thank Ms. Kathy Adkins, my supervisor at the Office of Business and Economic Research, College of Business, Oklahoma State University, for her support by employing me as a Graduate Research Assistant. I would also like to thank Mr. Tariq Hassan for the many tips and insights that he provided during all phases of this project.

Last but not the least, I would like to express my sincere gratitude to my parents, Mallikarjuna Rao and Saraswathi Rao, for their continued support and encouragement without which this endeavor would not have been successful.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

An important goal in the design of a computer system is to ensure that the design conforms to the requirements and the expected behavior of the system. The performance of an existing or proposed computer system can be evaluated or predicted by a number of techniques. A proposed system could be built and then evaluated based on experience from its actual use. Empirical data about the various system parameters can be gathered and performance measures of the system can be obtained. The performance of the system can also be measured by benchmarking the system with a predetermined set of jobs similar to those in the "real world".

The approach mentioned above entails a great deal of effort and cost. Alternately, a system could be modeled. Modeling a computer system is an important stage in its development. A system could be designed on the drawing board but its dynamic behavior is generally difficult to analyze. Modeling involves the construction of an abstraction of a system that reflects and responds to various system and job flow parameters. The model should also be able to exhibit performance measures that have a predictive value [Maekawa87].

One approach to modeling involves the software simulation of a proposed computer system with an elaborate internal structure representing the actual system being

modeled [Maekawa87]. Such models are accurate representations of the system and hence provide reliable information about the actual system performance. However, the construction of a software simulation model can be expensive and time consuming. It may also require a large number of executions with different sets of inputs to collect performance data of substance.

Another approach is analytical modeling of a computer system based on queueing theory [Maekawa87]. These techniques can be used to model complex computer systems. They have the advantage of being low in cost and flexible, but have the inherent disadvantage of being unable to truly reflect the detailed characteristics of a real system.

The popularity of modeling on computers has been made possible by the emergence of automated tools for different categories of problems. These tools could be used to depict and analyze computer systems. A complex computer system can be divided into logical sub-systems of manageable complexity, each of which can be modeled independently. The components of the sub-systems can be depicted using simple graphical shapes. Communication among sub-systems can be achieved and illustrated as needed. Each sub-system would need inputs which could result from other sub-systems. Using a top-down approach, it is possible to design and model the behavior of any sub-system assuming that the initial conditions are satisfied. As the next step in the modeling process, the model could be analyzed for conformity with the expected behavior and performance characteristics. The system design could then be refined based on the results of the analysis of the performance data collected. The iterative analysis and modification procedure can be utilized to construct a model with a desired behavior and performance characteristics. Finally, the sub-systems can be put together to model the

complete system, which can then be implemented.

In order to graphically represent a model, we can use a directed graph with nodes signifying various sub-systems or parts of a sub-system, and paths signifying interaction between different components of the system. The problem with this approach is that it is difficult to depict parallelism and synchronization among the different components of a system.

A queueing network [Gelenbe87] is a versatile modeling tool that can answer the problems mentioned above. A queueing network is an abstract model of the flow of jobs in a system. It is an analytical tool that can be applied to different classes of systems. A queueing network can describe a concurrent and/or asynchronous system. It can be analyzed for different performance characteristics such as throughput, turnaround time, and waiting time.

The main objective of this thesis was to develop a tool that can help in the development of a queueing network model of a system. Chapter II of this thesis provides a review of the current literature on queueing networks. Chapter III provides a discussion on the design and implementation details of the software that was developed as part of this thesis. The testing and evaluation of the software developed are discussed in Chapter IV. This thesis ends with Chapter V that provides a summary, the conclusions drawn from the study, and some suggestions for future work.

CHAPTER II

LITERATURE REVIEW

2.1 Definitions

This section contains some of the fundamental definitions about queueing networks. These definitions are mostly based on five major references [Lipsky77] [Trivedi82] [Kendall51] [Maekawa87] [Gelenbe87].

A *queueing network* is an interconnected collection of service centers in which jobs proceed from one center to another to satisfy their service requirements. Formally, a *queueing network* is a quadruple QN = (Q, S, O, P), where $Q = \{q_1, q_2, ..., q_n\}$ is a finite set of queues, each of some non-negative size, $S = \{s_1, s_2, ..., s_n\}$ is a finite set of servers, $O = \{o_1, o_2, ..., o_n\}$ is a finite set of environments or outsides, and $P \subseteq \{Q \text{ X } Q\}$ U { S X S} U {Q X S} U {S X Q} U {Q X O} U {O X Q} U {S X O} U {O X S} is a set of directed edges. Edges of type S X S can be used in modeling pipelined systems and edges of type Q X Q can be used in modeling subdivision of a gestalt queue into a number of queues based on priorities.

A *queue* in a queueing network is a node that is represented by an open rectangle. Each queue q has a preset and a postset. The *preset of a queue* can be defined as $*q = \{x \mid (x, q) \varepsilon P\}$, where *q represents the preset of queue q and x can be a queue, a server, or an outside. The *postset of a queue* can be defined as $q^* = \{x \mid (q, x) \varepsilon P\}$, where q*

represents the postset of queue q and x can be a queue, a server, or an outside. The interconnection of a queue with a set of queues, servers and outsides is defined by the preset and postset of that queue.

A *server* in a queueing network is a node that is represented by a circle. Each server s has a preset and a postset. The *preset of a server* can be defined as *s = {x | (x, s) ε P}, where *s represents the preset of server s and x can be a queue, a server, or an outside. The *postset of a server* can be defined as s* = {x | (s, x) ε P}, where s* represents the postset of server s and x can be a queue, a server, or an outside. The interconnection of a server with a set of queues, servers, and outsides is defined by the preset and postset of that server.

An *outside* or *environment* in a queueing network is a node that is represented by a square. Each outside o has a preset and a postset. The *preset of an outside* can be defined as *o = {x | (x, o) ε P}, where *o represents the preset of outside o and x can be a queue, a server, or an outside. The *postset of an outside* can be defined as o* = {x | (o, x) ε P}, where o* represents the postset of outside o and x can be a queue, a server, or an outside. The interconnection of an outside with a set of queues, servers, and outsides is defined by the preset and postset of that outside.

An *Open queueing network* is a queueing network that has one or more sources of job arrivals from outside the system and one or more sources of job departures to the outside of the system. A *Closed queueing network* is a queueing network that is characterized by jobs that circulate indefinitely within the network with no interaction with the outside of the system [Trivedi82].

The *Kendall notation* [Kendall51] is an abbreviation for the parameters that

describe a queueing system [Maekawa87]. A queue can be characterized with a notation of the form

A/B/c/k/m/Z

where

A = arrival process
B = service process
c = number of identical servers
k = queue capacity
m = customer population
Z = queueing discipline

The notation generally used for the arrival process or service process is one of the following [Gelenbe87].

GI = general independent distribution
G = general distribution
H_k = hyperexponential distribution of order k
E_k = Erlang distribution of order k
M = exponential distribution
D = constant distribution

The common queueing disciplines are

FIFO = first come, first served
LCFS = last come, first served
FIRO = first in, random out

An *arrival process* is one that is characterized by jobs that arrive at random independently of each other at times that are unknown. The length of the time interval between arrivals is a random variable, taking values from one of the distributions mentioned above, with an average of λ [Lipsky92].

A *service process* is one that is characterized by one or more identical servers serving jobs in a queue. The service time is a random variable, taking values from one of the distributions mentioned above, with an average of μ.

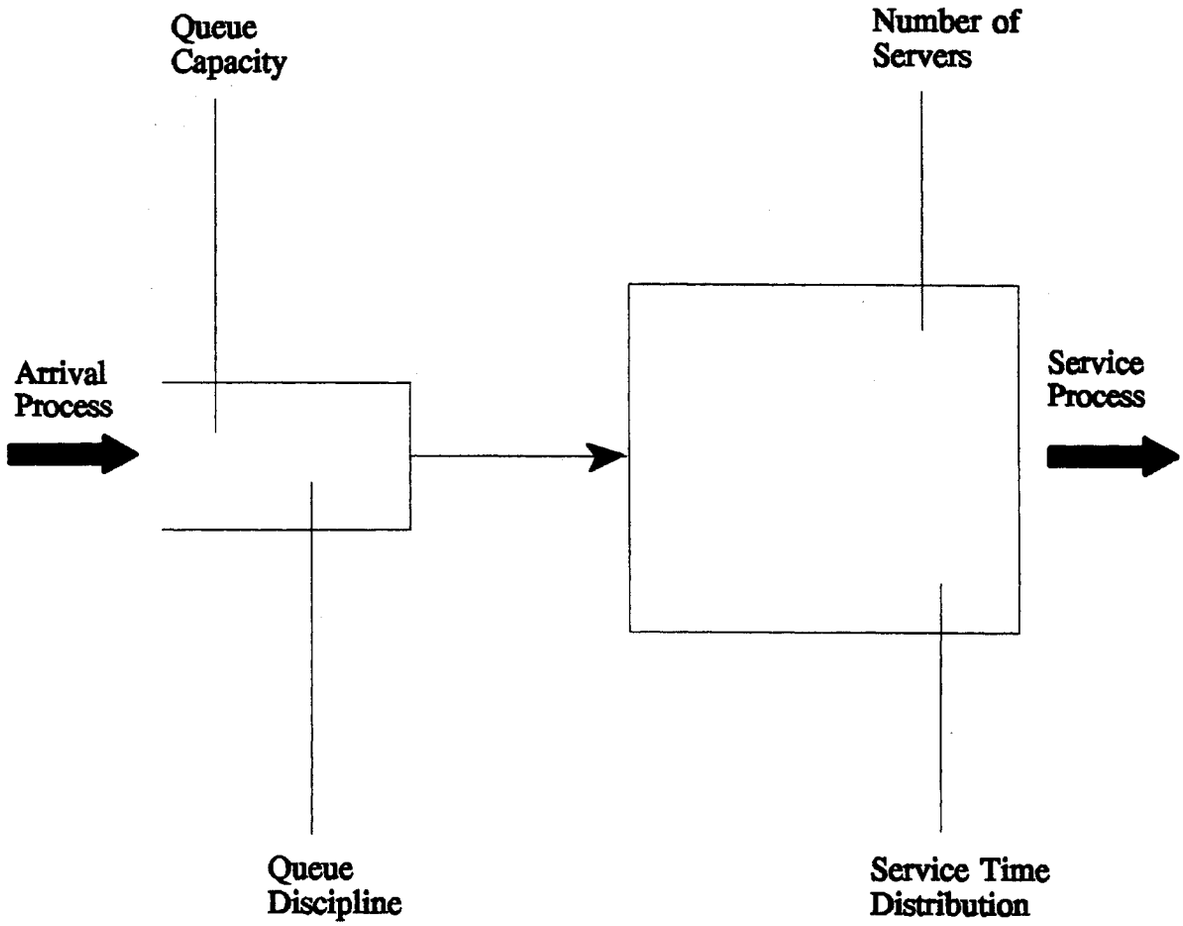

Figure 1. Parameters of a queue

The *common performance measures* used in evaluating a computer system are listed below.

- *Mean queue length*, $\bar{n}$, is the average number of jobs in the queue including the one being served at the steady state.
- *Mean waiting line length*, $\bar{w}$, is the average number of jobs in the system that are not being served at the steady state.
- *Mean waiting time*, $\bar{t}_w$, is the average time a job waits for service excluding the service time at the steady state.
- *Mean turnaround time*, $\bar{t}_t$, is the average time a job spends in the system from the

time it enters the system to the time it exits the system. It is the sum of the mean service time and the mean waiting time.

- *Processor utilization,* ρ, is the ratio of λ to μ or the ratio of the average arrival rate to the average departure rate.

A queueing network can be defined as a *tripartite graph* with three types of nodes: circles representing servers, open rectangles representing queues, and squares representing outsides or environments. The connection between queues, servers, and outsides is through directed paths. Queues directly connected together, servers directly connected together, or outsides directly connected together can be considered as a single queue, a single server, or a single outside respectively. Hence, a queueing network can be considered as a tripartite graph. A sample queueing network is given in Figure 2. The queueing network can be defined as QN = (Q, S, O, P), where Q = $\{q_1, q_2, q_3\}$, S = $\{s_1, s_2, s_3\}$, O = $\{o_1, o_2\}$, and P = $\{(s_1, q_1), (s_1, q_2), (s_1, q_3), (q_1, s_1), (q_2, s_2), (q_3, s_3), (s_2, q_1), (s_3, q_1), (o_1, q_1), (s_1, o_2)\}$.

2.2. Characterization of Queueing Networks

Modeling computer systems involves dealing with multiple resources such as CPU's, memories, channels, and disks. The complexity of modeling computer systems leads to the use of queueing networks rather than a single queue with a single server. However, an understanding of the single queue model is necessary for the following reasons [Gelenbe87].

- Understanding queueing theory is easier when a model with a single queue with a single model is considered.
- It is a useful framework for the development of mathematical tools for analyzing complex queueing networks.

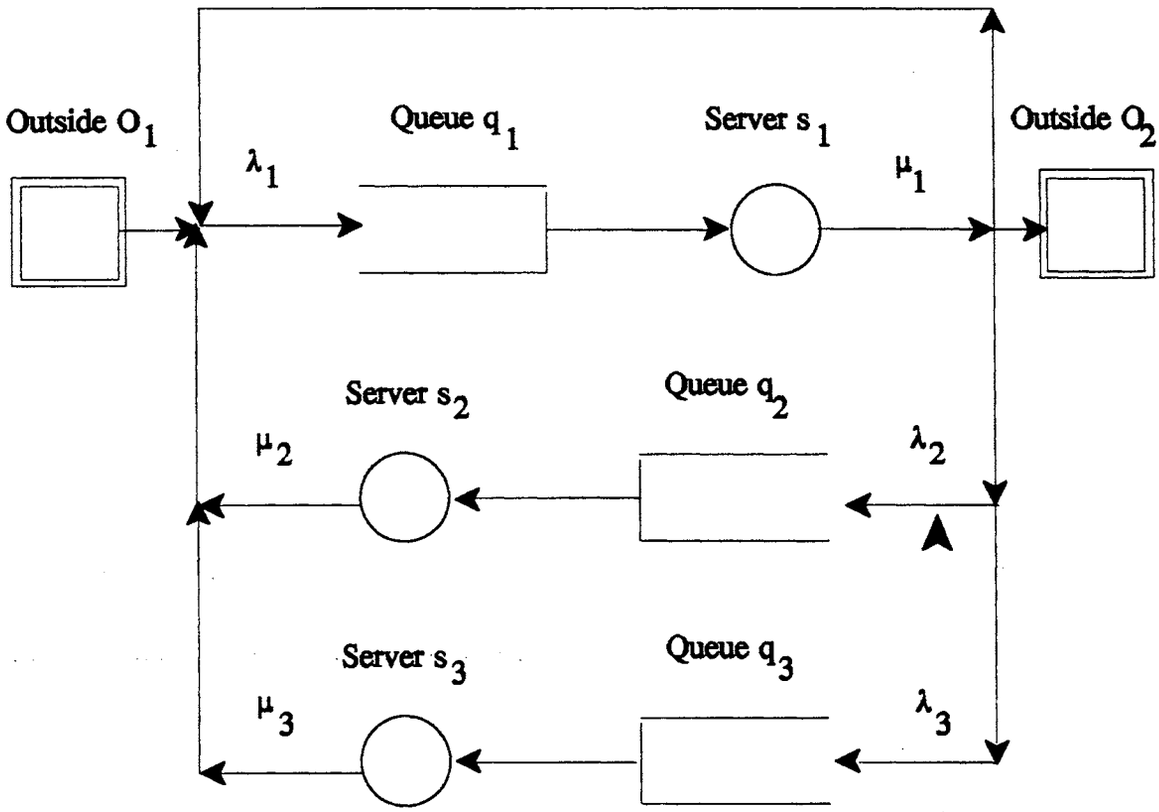

Figure 2. An open queueing network for a simple computer system

- The entire computer system can be considered as an unique server with a complex queue discipline.

The behavior of a system can be analyzed using queueing theory. The behavior of such a system can be categorized as follows [Maekawa87].

- The system characteristics are based on its steady state behavior, i.e., the probabilities and distributions are invariant with respect to time.
- The processes in the system are stochastic with probabilistic distributions and are independent of each other. The stochastic behavior of the processes is obtained from empirical studies of real systems.
- The population of jobs requesting service is either infinite or finite.
- The jobs follow the *Markov process* as they move within the system.

- The distribution of service times is exponential.
- The capacity of the individual queues in the system is either finite or infinite.
- The average performance characteristics of the system converge to those at steady state, i.e., it is an *ergodic system*.

2.2.1 Erlang's Model

A basic model for studying queueing systems is Erlang's model. Figure 3 provides a queueing model for a uniprocessor multiprogrammed system.

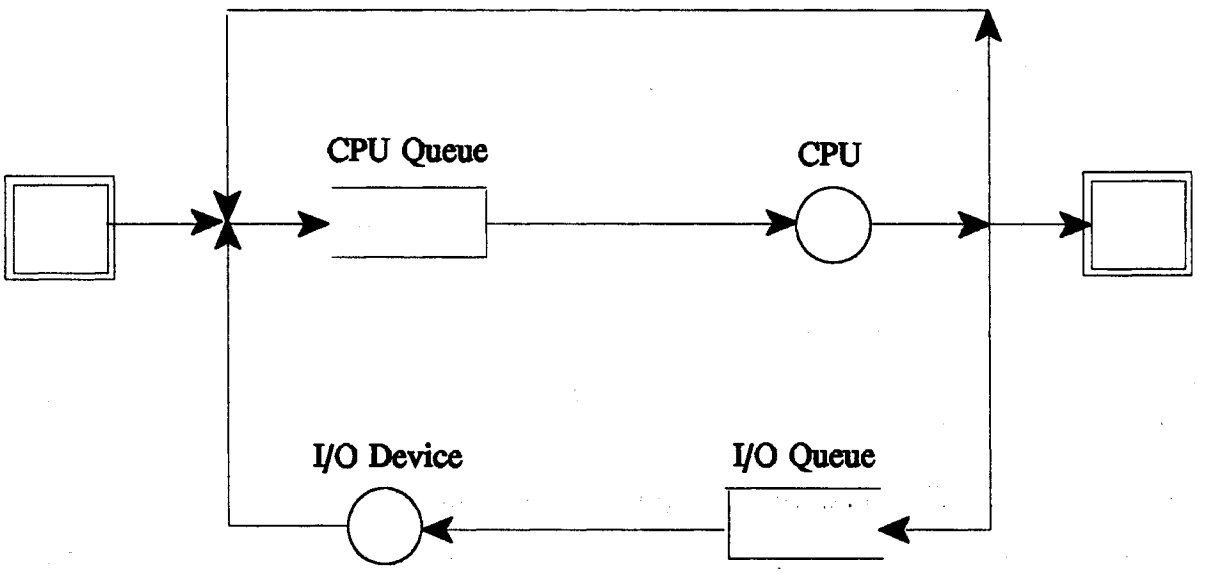

Figure 3. A queueing model for a uniprocessor multiprogrammed system

Based on Figure 3, *Erlang's model* suggests that any queueing model consists of three components.

- Servers are the centers where a job receives service. The individual servers are

characterized by a *time distribution of service*. This represents a probability distribution function that describes the time required to service a job. The mean service time is μ. The most common probability distribution function for a service process is exponential since it most closely represents a real computer system [Maekawa87].

- Queues are the centers where jobs line up for service. The individual queues are characterized by a queue capacity, arrival process, and a queue discipline. The *queue capacity* may either be finite or infinite. The arrival process is one in which jobs arrive at intermittent times from within or outside the network. The precise time of arrival of these jobs is unknown. The arrival process continues indefinitely. The arrival process can be represented by a probability distribution function that describes the number of jobs arriving in an interval of time. The mean rate of arrival of jobs into a queue is represented by λ. The most common probability distribution function for an arrival process is the Poisson distribution since it provides an accurate representation of the rate of arrival of jobs in a computer system. The *queue discipline* represents the order in which jobs in a queue are serviced by one or more servers. The most common queue discipline is first-come, first-served due to its inherent simplicity in analyzing computer systems.
- *Paths* are directed arcs between nodes in a queueing network. Each path is characterized by a probability value γ that a job moves along it.

2.2.2 Poisson Distribution

The distribution of the arrival of jobs for most systems follows a Poisson

distribution. A *Poisson distribution* is characterized as follows [Maekawa87].

- The number of jobs arriving in an interval of time is a random number taken from a probability distribution function.
- In a "short" interval of time, the probability of one arrival is proportional to the length of that time interval, Δt, or

  $$\mathbf{P\ [1\ arrival\ in\ time\ \Delta t] = \lambda \Delta t}$$

  where λ is the mean rate of arrival of the jobs.
- The probability of more than one arrival in a "short" interval of time is negligible, i.e., the chance of a spike in the distribution is very low, or

  $$\mathbf{P\ [more\ than\ 1\ arrival\ in\ time\ \Delta t] = O(\Delta t)}$$

  The property of $O(\Delta t)$ can be mathematically expressed as follows: any function of Δt, $f(\Delta t)$, has the $O(\Delta t)$ property if

  $$\lim_{\Delta t \to 0} \frac{f(\Delta t)}{\Delta t} = 0$$

- The Poisson distribution is a discrete distribution. The probability of exactly n jobs arriving in an interval of time Δt is given by

  $$P\ [N = n] = \frac{e^{-\lambda \Delta t}(\lambda \Delta t)^n}{n!}$$

  where λ is the mean rate of arrival of jobs.

2.2.3 Distribution of Interarrival Times

The *distribution of interarrival times* is related to the Poisson distribution of arrivals and has the following properties.

- The distribution of interarrival times is an exponential distribution.
- The probability distribution function for the interarrival times can be expressed as

$$P\,[T \le t] \;=\; 1 - e^{-\lambda \Delta t}$$

where λ is the mean rate of arrival of jobs. The mean of this distribution is $1/\lambda$ and the variance of this distribution is $1/\lambda^2$.

- The exponential distribution is *memoryless*, i.e., the number of arrivals in one state of the system does not depend on the previous state(s). This implies that, after a sufficient length of time, the initial transient behavior of the system will be forgotten and systems with an exponential interarrival time distribution will approach their steady state.
- Every Poisson arrival process has an exponential distribution of interarrival times.

2.2.4 Aggregation and Branching of Paths

An important issue in queueing networks is the branching and aggregation of paths. Consider the *aggregation of paths* into one path as depicted in Figure 4. Specifically, there are n paths that are merging into one path. The n paths represent Poisson processes with mean rates of λ_1, λ_2, ..., or λ_n. There is one "merged" path with a mean departure rate of λ. The relation between λ and λ_1, λ_2, ..., λ_n is

$$\lambda \;=\; \sum_{i=1}^{n} \lambda_i$$

Further, the resulting aggregate path is also a Poisson stream.

The *branching of paths* is depicted in Figure 5. Specifically, there is one path that

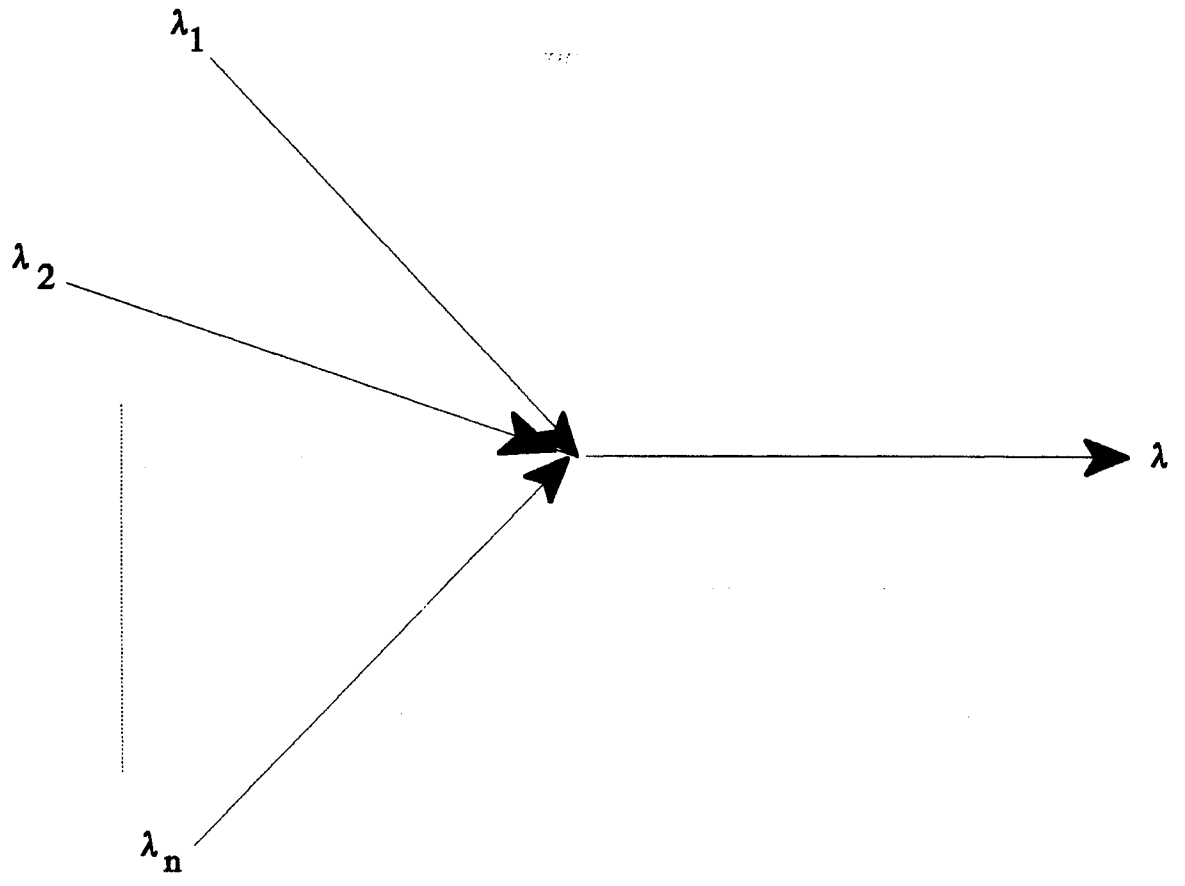

Figure 4. Aggregation of paths

has a Poisson distribution with a mean rate of λ. To receive service, a job must choose one of the n branches with rates λ_1, λ_2, ..., or λ_n. The probability of joining any of the n paths is γ_1, γ_2, ..., or $\gamma_n = 1 - (\gamma_1 + \gamma_2 + ... + \gamma_{n-1})$. The relation between λ and λ_1, λ_2, ..., λ_n is the following, for i = 1 to n,

$$\lambda_i = \gamma_i \lambda$$

Each of the resulting output paths is also a Poisson stream.

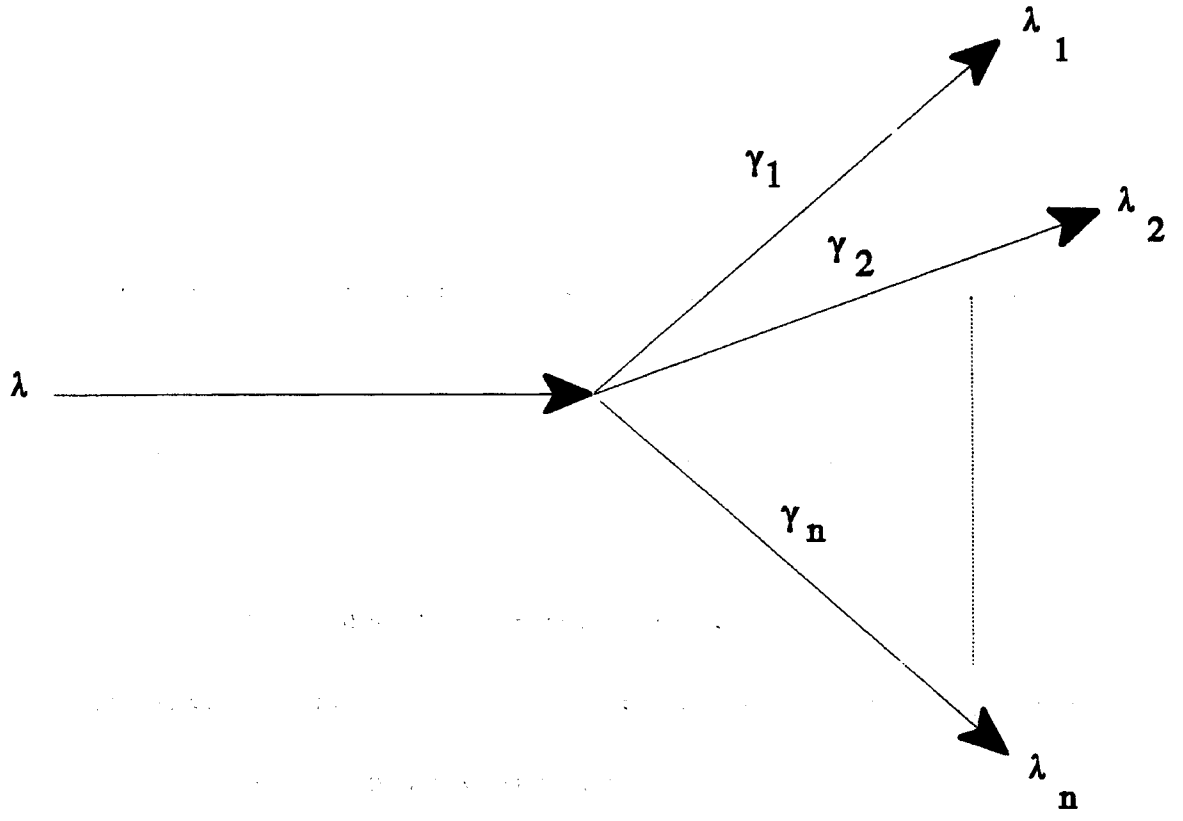

Figure 5. Branching of paths

2.2.5 Analysis of an M/M/1 System

The behavior and performance characteristics of an M/M/1 system without the details of the derivation process are considered in this subsection. Figure 6 depicts an M/M/1 queue without feedback.

Mean Value Analysis is used extensively while analyzing queueing networks. The *mean value analysis* of an M/M/1 queueing model is characterized as follows.

- A Poisson arrival process with a mean arrival rate of λ jobs per unit time.
- An exponential interarrival time distribution with an average of $1/\lambda$ time units.
- An exponential service time distribution with a mean service time of μ time units.

Figure 6. An M/M/1 queueing model without feedback

- An average service rate (or departure rate) of $1/\mu$ jobs per unit time.

A steady state system is a necessity for accurate prediction of the behavior of a system using mean value analysis. It is to be noted that being in the steady state does not imply that the system is constant.

Little's law states that the average queue length, $\overline{n}$, is proportional to the average waiting time for a job $\overline{t_w}$ [Little61].

$$\overline{n} = \lambda \overline{t_w}$$

The system utilization, ρ, is the product of the mean arrival rate and the mean service rate, or

$$\rho = \frac{\lambda}{\mu}$$

Utilization is an important measure of the performance of a system. If $\rho > 1$, the system will never reach the steady state since there are more jobs arriving into the system than the server can handle, and the queue length will grow indefinitely. If $\rho < 1$, the system will attain the steady state quickly. Further, ρ is the percentage of time the server remains busy. If ρ is close to 1, the system may eventually reach the steady state [Maekawa87]. Utilization is an important measure since we desire that the system be

stable and reach the steady state quickly.

The *steady state queue length probability*, S_n, is the probability that there are exactly n jobs in the queue at the steady state. It has been shown that at the steady state we have [Maekawa87]

$$S_n = \rho^n (1 - \rho), \quad n \geq 0$$

assuming that $\rho < 1$ and $\sum_{n=0}^{\infty} S_n = 1$

The expected number of jobs in the system, $\bar{n}$, including the job being served is

$$\bar{n} = \frac{\rho}{1 - \rho}$$

The mean queue length, $\bar{w}$, is

$$\bar{w} = \frac{\rho^2}{1 - \rho}$$

The average waiting time for a job, $\bar{t}_w$, is

$$\bar{t}_w = \frac{\rho}{\mu - \lambda}$$

The average service time for a job, $\bar{t}_s$, is

$$\bar{t}_s = \frac{1}{\mu}$$

The average turnaround time for a job, $\bar{t}_t$, is

$$\bar{t}_t = \frac{1}{\mu - \lambda}$$

2.3. Modeling with Queueing Networks

Queueing networks are commonly used to model different systems. Queueing networks capture the potential parallelism effectively. The open queueing network shown in Figure 7 is a model of a terminal-oriented distributed computing system [Trivedi82].

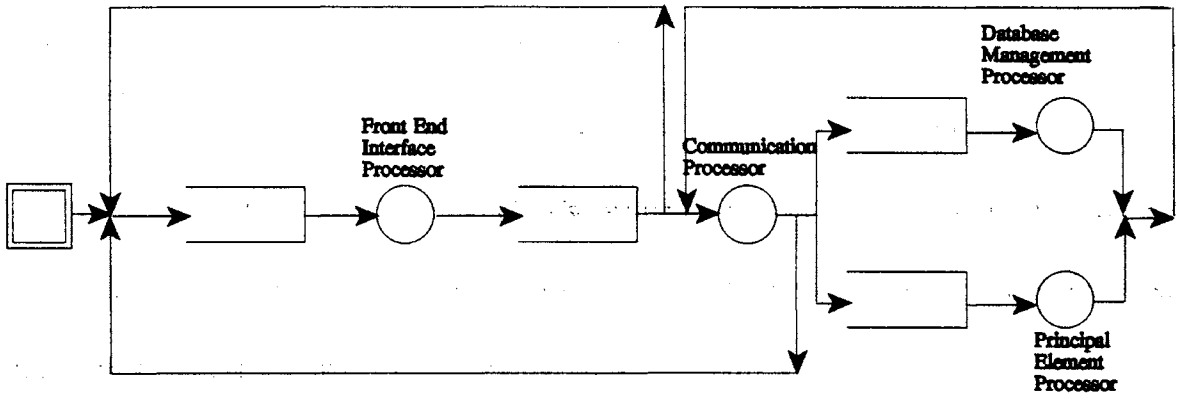

Figure 7. An open queueing network model of a terminal-oriented distributed computing system

A multiple bus configuration computer system can be represented by a closed queueing network [Lavenberg88]. In this class of queueing networks, jobs never enter or leave the system.

Queueing networks can represent different computing systems such as time sharing systems [Klienrock64], round robin scheduling [Klienrock76], machine repair model of a time sharing system [Scherr67], multiprogramming systems [Gaver67] [Lewis71], central server model [Jackson63] [Gordon67], I/O sub-systems [Fuller75], priority queues [Byrant84], exclusive and shared locks [Mitra85], asynchronous parallelism [Heidelberger82] [Thomasian86], fork/join parallelism [Heidelberger83], and load sharing

systems [Wang85] and flow of control in computer programs containing different constructs such as DO-WHILE, IF-THEN-ELSE, CASE, and PARBEGIN-PAREND.

2.4. Issues in Queueing Networks

In this section, some of the issues in modeling complex computer systems are briefly considered.

2.4.1 Single Queueing Systems

There are several versions of the M/M/1 queueing system such as M/M/c, M/M/∞, M/G/1, and M/GI/∞. These systems have been analyzed in a manner analogous to that presented in Section 2.2.5 for an M/M/1 queue.

2.4.2 Open Network of Queues

When modeling a central server computer system, it is necessary to develop the model as an open network of queues since any job submitted by a user to the system can be considered as entering from outside the system [Maekawa87]. In such a network, there are K queues. Each queue is characterized by feeding many identical servers and an arrival process that can be described by a Poisson distribution with an average rate of arrival of λ_m. A job from queue i, after being serviced, joins queue j with the probability of γ_{ij}. The routing probabilities among the queues can be represented by a K x K matrix, where K is the total number of queues. A job from queue j leaves the system with probability [Maekawa87]

$$1 - \sum_{k=1}^{K} \gamma_{ij}$$

A requirement for this type of network is that the network must be totally interconnected and the K x K matrix cannot have all the rows summing to one. This is to ensure that jobs leave the system. Jackson obtained a solution for an open network of queues [Jackson57]. Jackson's solution assumes that all arrivals (from outside the system and from one queue to another) are Poisson distributed with the service times being exponentially distributed. Later, Jackson demonstrated a solution that removed this latter restriction [Jackson63]. The elegant nature of these solutions is the fact that the performance results for open networks can be derived from those for an M/M/c queue.

2.4.3 Closed Network of Queues

When modeling interactive systems that have a fixed number of terminals, each with an user entering commands requiring processing by the CPU and a combination of one or more devices, we need to resort to a closed network of queues [Maekawa87]. In a closed network of queues, there are no arrivals from or departures to the outside of the system. The total number of jobs in the system is constant at $\bar{n}$. A closed network is similar to an open network except that:

- The sum of each of the rows in the *product-form matrix* (a square matrix of size K, the number of queues in the system, in which each element of the matrix indicates the probability that a job in a queue i after being served joins queue j) is 1 since there are no departures from the system.

- The service rate, in general, depends on size of the queue(s) that feed it.

The analysis of closed queueing networks can be performed by using the two different techniques of normalization constants and mean value analysis as outlined below [Maekawa87].

As the name implies, the *normalization constant method* requires the computation of a normalization constant to ensure that the steady state probability of each state sums to one. Unfortunately, this computation requires exponential time which is not practical [Maekawa87]. Buzen developed an algorithm that can compute the constant in polynomial time [Buzen72] . There are well-known techniques that permit one to compute the queue length probabilities for *load dependent* and *load independent* systems along with the performance measures for these systems [Maekawa87].

The *mean value analysis* technique computes the performance measures without requiring the time intensive computations of the normalization constant method. However, it has a higher space complexity [Maekawa87].

Finally, certain restrictions on closed queueing networks can be removed to analyze generalized networks known as *BCMP* (Baskett - Chandy - Mains - Palacios) networks [Baskett75]. These generalizations are listed below [Maekawa87].

- There are multiple job classes and each of these job classes may have a different service time. These jobs may change classes and different job classes may have different routing probabilities.
- Some jobs may come from outside the system while some may not. Thus, jobs can fall into open and closed network classes and may also switch classes.
- There may be many queueing disciplines in the system.

- General service time distributions are permitted.

Such queues have been analyzed using the normalization constant method, which requires an exponential computation time [Baskett75]. A *tree convolution method* was presented by Lam and Lien that is computationally efficient both in terms of time complexity and space complexity [Lam83]. BCMP networks have also been analyzed using the mean value analysis method [Bruell80] [Zahorjan81]. The tree convolution method mentioned earlier has also been adapted for the mean value analysis technique [Hoyme68].

CHAPTER III

IMPLEMENTATION ISSUES

3.1 Implementation Platform and Environment

3.1.1 Sequent Symmetry S/81

The Sequent Symmetry S/81 is a mainframe class computer system with a multiprocessor architecture that was developed by Sequent Computer System, Inc. The multiprocessing and shared memory architecture consists of the following elements [Sequent90]:

- A parallel architecture that utilizes multiple industry-standard microprocessors.
- Either the DYNIX v3.0 operating system or the DYNIX/ptx operating system. Both options are UNIX system ports.
- A standard set of interfaces to the network such as Ethernet, SCSI, VMEbus, and MULTIBUS.

The operating system of the Sequent Symmetry S/81 has been engineered to incorporate features that support the parallel architecture. In addition, software that has been built for the UNIX operating system can run on the Sequent Symmetry S/81 with little or no modification. In the case of multi-user applications, the operating system of the Sequent Symmetry S/81 automatically distributes the tasks to multiple processors in an attempt to reduce response time and increase system throughput [Sequent90].

DYNIX v3.0 operating system supports the two major command sets of UNIX, namely, the Berkeley UNIX and UNIX System V. On the other hand, the DYNIX/ptx operating system is compatible with AT&T System V v3.2 only [Sequent90].

3.1.2 The X Window System

The X Window System is a software environment that was used to develop the graphical user interface (GUI) for the queueing network design tool developed for this thesis research. X's device independent graphics permit software developers to engineer portable GUIs [Young90]. The only requirement for complete portability is that the X protocol should be supported by the hardware platforms to which the software is to be ported. The interaction between a client and a server is defined by the X protocol. The X Window System follows a client-server architecture. An application acts as a client and the responsibility for all input and output devices is with the server [Young90].

The interaction between an application and the X Window System is provided by the X library. An example of a library of C language functions, which provide the user access to the device independent graphics of X and interface routines, is Xlib [Johnson90] [Bakrabati91] [Keller90]. Toolkits are easier to use than Xlib for developing GUIs. The standard toolkit for the X Window System is the X Toolkit. It consists of two modules: The Xt Intrinsics, which is a layer that directly interacts with the X Window System, and the widgets, which are a set of user interface building blocks [Johnson90]. Many popular widgets are supported by Xt Intrinsics. One of the more popular widget sets is the Motif widget set by Open Software Foundation (OSF). The Motif widget set supplies the GUI components such as windows, menus, buttons, scroll bars, icons, and bitmaps. The

relationship between the various layers in the X Window System is shown in Figure 8.

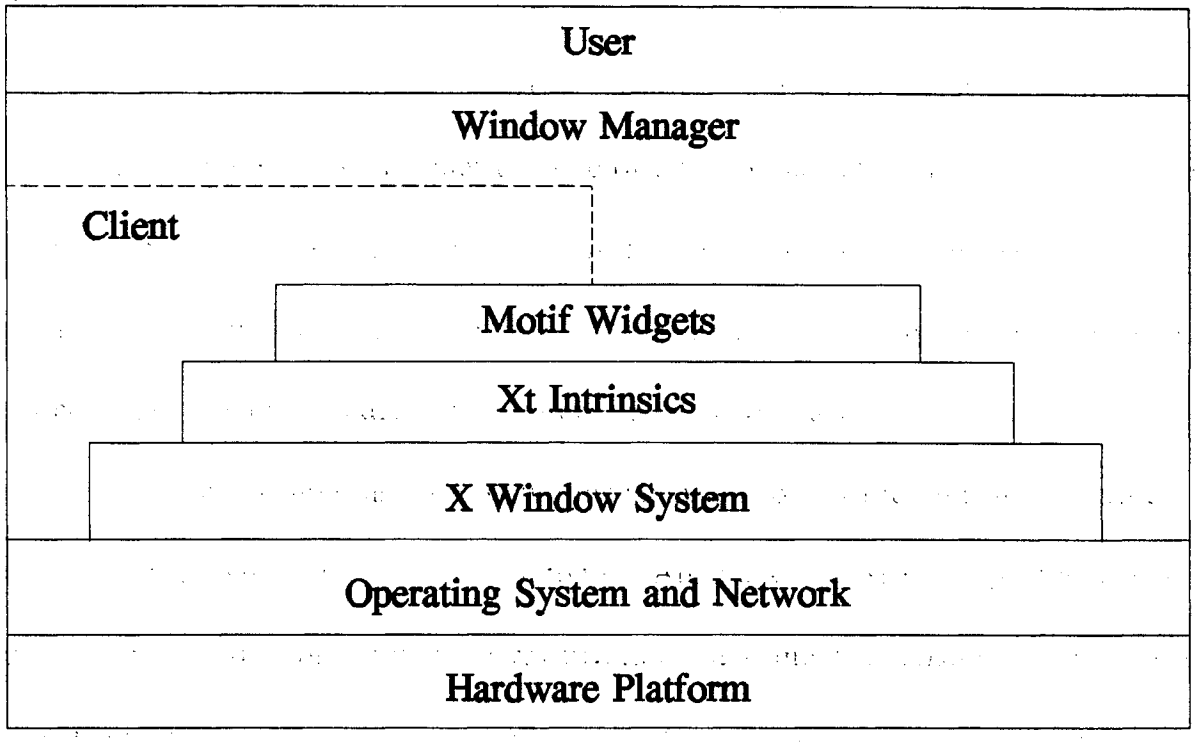

Figure 8. Layers in the X Window System

3.1.3 OSF/Motif Toolkit

The OSF/Motif Toolkit was designed by Open Software Foundation (OSF) and is based on the X Toolkit Intrinsics (Xt). The OSF/Motif Toolkit is a set of functions and procedures that provides quick and easy access to the lower layers of the X Window System. The OSF/Motif functions and procedures provide user-interface objects known as widgets. OSF/Motif is a specification rather than an implementation, making it entirely implementation independent [Heller91].

The complete architecture of the OSF/Motif Toolkit consists of a number of important modules that are shown in Figure 9 [Berlage91]. From Figure 9, it is apparent

that the hardware independent nature of the X Window System stems from its acting as the lower layer. The primary window functions such as resizing, closing, moving, and iconizing are managed by the Motif Window Manager (MWM). In achieving its tasks, the Motif Window Manager follows the Inter-Client Communication Conventions (ICCC) that enable it to manage X applications developed using different toolkits [Berlage91]. The Motif Window Manger also manages other functions such as controlling the input focus to determine which application window should receive input and maintaining the stacking order of overlapping windows [Berlage91].

The presentation details of the user-interface elements are specified in the User Interface Language (UIL). The UIL compiler translates the presentation details specified and loads it into memory at run time. However, the development of a Motif application does not require the UIL component [Berlage91].

The Motif Toolkit which provides a set of widgets is the most important module of the OSF/Motif architecture shown in Figure 9. The Motif Toolkit provides widgets for such common user-interface objects such as push buttons, menus, labels, dialog boxes, scroll bars, and text entry or display areas. In addition, there are widgets known as managers that perform the function of controlling the layout of other widgets. A widget operates, to a large extent, independently of the application. A widget's actions are determined by the events dispatched to it by the Xt Intrinsics. For example, a cascade button knows how to draw itself, how to highlight itself, and how to respond to a mouse click (or an user-defined action) by executing an application procedure [Nye90] [Heller91].

The class inheritance hierarchy of the Motif widget set is shown in Figure 10.

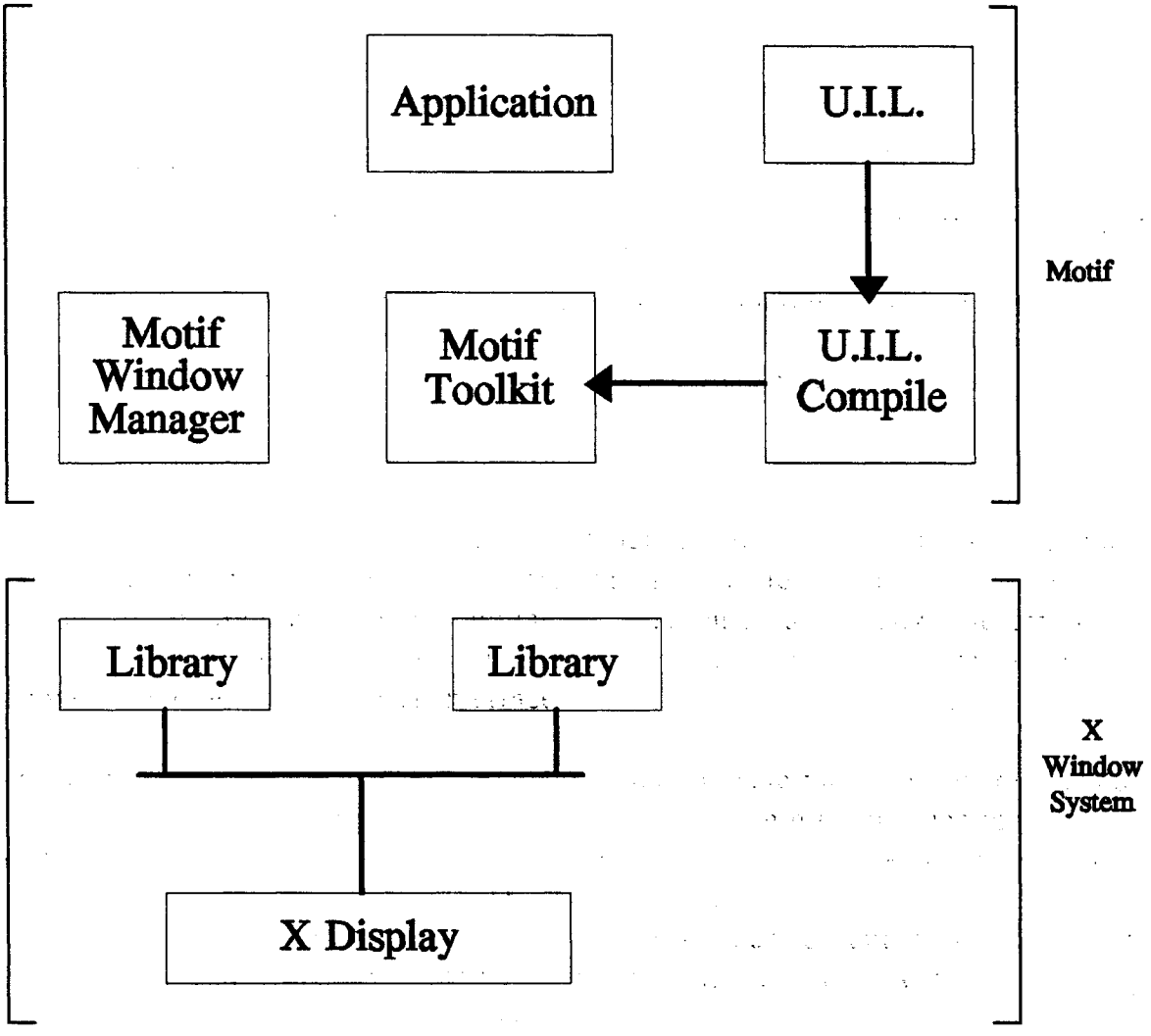

Figure 9. Architecture of OSF/Motif (Source: [Berlage91])

Some of the classes shown in Figure 10 are defined by the Xt Intrinsics. Some of the base classes in the class inheritance hierarchy are defined by the Xt Intrinsics. Their behavior can be inherited by widgets that are derived directly from the Xt Intrinsics defined base classes. They also provide a common behavior for all widget classes based on them.

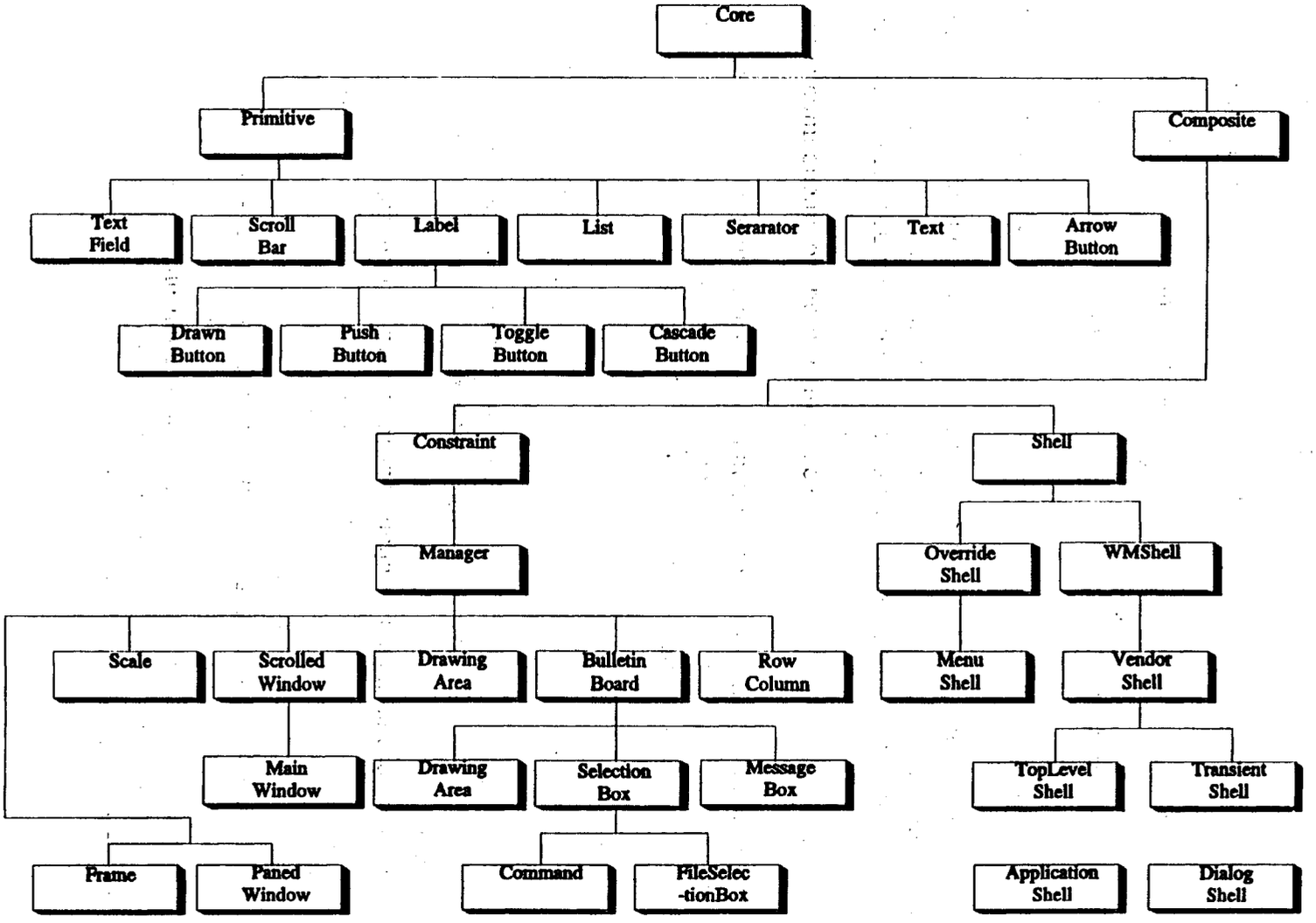

Figure 10. A class inheritance hierarchy for the Motif widget set (Source: [Nye90])

The root of the widget class hierarchy is the Xt Intrinsics Core widget class. It is the super-class for all widget classes derived from it, and provides a set of common resources such as the size and the position that is inherited by all the other classes.

The Primitive widget class is the highest level Motif widget class. It is derived from the Core widget class. It inherits some of its resources from the Core widget class and adds some of its own resources such as control of the three-dimensional shadows. The Label widget class is derived from the Primitive widget class and hence inherits some resources from it while it adds some features of its own, such as its ability to display a pixmap or a string of text, as well as the mechanisms for positioning of the string and displaying of the string in a variety of fonts. All the sub-classes of the Label widget class, i.e., the PushButton class, the DrawnButton class, the ToggleButton class, and the CascadeButton class, inherit features together with the resources added by them.

The Composite widget class is an Xt Intrinsics widget class that is sub-classed from the Core widget class. It adds features that provide it the capability to manage its geometry. The Constraint widget class is also an Xt Intrinsics widget class that is derived from the Composite widget class and is a refinement upon the Composite widget class. It provides the user or the application a method to manage the position and size of the widget. The Manager widget class provided by Motif is sub-classed from the Constraint widget class. It is the super-class for all the widgets that can manage the geometry of their children such as the RowColumn widget class, the DrawingArea widget class, the Scale widget class, and the BulletinBoard widget class [Nye90].

The Shell widget class is an Xt Intrinsics widget class that is a sub-class of the Composite widget class. Shell widgets provide an interface between the window manager

and other widgets. The functions of the Shell widget class are to handle the window manager protocol for the application and to set the resources required by the window manager. There are a number of Shell widget classes since the function of interacting with the window manager is very complex. The OverrideShell widget class is derived from the Shell widget class and provides a temporary window that completely bypasses interaction with the window manager. The MenuShell widget class is derived from the OverrideShell widget class and was introduced by Motif to handle the special interface requirements of the OSF/Motif architecture. The WMShell widget class that is sub-classed from the Shell widget class is a set of simple, wire bed-frame widgets that has no special attributes. The VendorShell widget class is sub-classed from the WMShell widget class that vendors can use to define their own attributes that are specific to their own window managers. The TransientShell widget class is used by the Xt Intrinsics to create dialog boxes. The TransientShell widgets may not be iconified separately by the window manager. However, if an application is iconified, all the child widgets of the application that belong to the TransientShell Widget class are automatically iconified by the window manager. The DialogShell widget class is sub-classed from the TransientShell widget class and was created by OSF/Motif. The functions of the TopLevelShell widget class and the ApplicationShell widget class provide various applications with their top-level windows [Nye90] [Young90] [Heller91].

3.2 Implementation

3.2.1 Program Structure

The code for QNT has been divided into 22 C program modules. Three modules

of the program are used to set up the initial environment of QNT. The remaining modules are used to implement the various options provided by QNT. Each module includes functions necessary to implement one or more options available in the software tool. This method of modularization has been used so that modifications to the functionality of the program can be restricted to one module and can be performed independently of the remaining modules. The different modules of the program are briefly described below.

- main.c: This module of QNT is the main program file. The initial user interface of QNT is set up by the various external functions called from this module. The functions in the program module "initialize.c" are called to initialize the environment variables used in QNT. The functions in the module "gui.c" are used to build the various components of the user interface.

- allocate.c: This module consists of functions that allocate the memory for a queue, a server, an outside (or environment node), or a path. There is one function to allocate memory for a QUEUE data structure that contains all the relevant data for manipulating a queue. This function also initializes the various fields of the data structure QUEUE. Similarly, there is one function each to handle the data structure SERVER, the data structure OUTSIDE, and the data structure PATH.

- analyze.c: The functions in this module implement the "Analyze" option in the tool box. This module consists of functions to display the dialog box used to prompt the user for the subnet to be analyzed, check if it is a valid subnet, display a dialog box for the user to edit the parameters of the subnet selected, and display in a dialog box the steady state system characteristics of the subgraph being analyzed.

- check.c: This module consists of the functions necessary to check and validate the queueing network drawn by a user. It checks to ensure that the constraints of a queueing network are satisfied. If certain nodes do not satisfy the constraints, they are listed by the type of the node in a dialog box. In addition, this module consists of functions that enable the user to select a node that is not satisfying a constraint and edit its characteristics to bring it within the limits of the definition of a queueing network.

- close.c: This module consists of functions necessary to implement the "Quit" and "Close" options in the "File" pull down menu. Among its many responsibilities, it provides an "Exit" dialog box as a safety measure when the "Quit" option is selected, to verify if the user really wants to exit from the application. It also contains functions to display a dialog box that prompts the user to save the file, when the user selects the

"Close" option from the "File" pull down menu or selects "Close" from the system menu associated with the dialog shell of the canvas or the top-level shell of the application.

- cursors.c: The functions in this module create the different cursor shapes, assign one of them as the shape of the current cursor for a particular window, and reset the shape of the cursor if required. The cursor shape used depends not only on the action selected but also the window with which the cursor is associated.

- delete.c: This module comprises functions that are necessary to implement the "Delete Element" option in the "Edit" pull down menu. It primarily consists of two functions, one for selecting the object to be deleted and the other for deleting the selected object.

- draw.c: This module consists of functions that aid in drawing the queue, server, outside (or environment node), path including the arrow head, and labels. These functions make use of the different Xlib drawing routines such as XDrawSegments, XDrawLine, XDrawPolyLine, XDrawArc, XDrawString, and XDrawFilledArc.

- expand.c: This module consists of functions necessary to expand a node in any level of the network to the next higher level. If the higher level already exists, the queueing network at this level is displayed in a new canvas on top of the existing dialog shells. Otherwise, an empty drawing area is displayed within a dialog shell. The file name is displayed as the title of the window in which the canvas is displayed. If the file name has not already been provided, it is displayed as "Untitled".

- gui.c: This module consists of functions that are called from the module "main.c" for establishing the initial user interface. The elements of the initial user interface consist of pull down menus, a window for displaying the current action selected, a panel for the toolbox, and an empty area where the logo for QNT is displayed. When the user selects the "New" option or the "Open ..." option from the "File" pull down menu, a canvas is displayed in a dialog box on top of the logo for drawing the queueing network. The file name is displayed as the title of the window in which the canvas appears. Since initially no action is selected, the current action selected is displayed as "None".

- help.c: This file contains the various help messages associated with each option in the "Help" pull down menu. It also contains a function to display the "Help" dialog box with the appropriate help message in it.

- initialize.c: This module initializes the various global variables and data structures used in QNT. To perform the initialization, this program file implements different functions that are called from the module "main.c". The functions declared in this module also create the different cursors used by the software tool.

- jobs.c: This module consists of functions necessary to implement the "Place Jobs" option in the toolbox. When this option is selected, a dialog box is displayed to

prompt the user for information. The dialog box management and other operations associated with the "Place Jobs" option are handled by the functions in this module.

- label.c: The functions in this module implement the "Edit Label" option from the "Edit" pull down menu. This option, when selected, displays a dialog box that prompts the user for the old label that is to be changed as well as for the new label. The functions in this module also check to ensure that the old label exists and is valid. It also ensures that the new label is unique.

- messages.c: This file contains the functions for processing the different error codes generated by QNT. It also displays an error dialog box along with the appropriate error message in it.

- open.c: This module consists of functions necessary to implement the "New" and "Open ..." options from the "File" pull down menu. The file selection for opening a file is done through the use of the FileSelectionDialogBox widget provided by Motif. This module also contains functions to read and display an existing file on the canvas. The file name is displayed as the title of the window in which the canvas is displayed. If the file name has not already been provided, it is displayed as "Untitled".

- path.c: The functions necessary to draw a path are included in this module. When the "Path" option is selected from the toolbox, a function in this module is called to display a dialog box. It prompts the user for the label of the source and destination nodes between which the path is to be drawn. The module also includes functions to check the validity of establishing a path from the source node to the destination node.

- print.c: This module consists of functions necessary to implement the "Print ..." option in the "File" pull down menu and the "Dump ..." option in the "Performance Characteristics" dialog box. The former option creates a postscript image of the QNT window and the queueing network in the drawing area and saves it to a file. The latter creates a text file of the performance characteristics of the system being analyzed by the user.

- save.c: This module consists of functions necessary to implement the "Save" and "Save As ..." options in the "File" pull down menu. It also contains functions to write the information, pertaining to the drawing displayed in the canvas, to a file.

- specification.c: The functions in this module implement the "Show Specification" and "Edit Specification" options in the toolbox. They create a dialog box and provide the user with an editable text area to describe the interpretation of the node or path, and enter the parameters of the node or path. One function is also provided to decide which node or path has been selected by the user.

- undo.c: The functions in this module implement the "Undo" option provided in the "Edit" pull down menu. Alternately, the "Undo" operation can be selected by clicking the third button of the mouse with the cursor placed inside the canvas.

- utilities.c: This module includes various functions that are used by the software tool such as those for encoding a file, decoding a file, displaying the action selected by the user in the message bar, and redrawing the queueing network in the canvas when it is uncovered by another window.

Each module contains a header file. For the purpose of consistency, each header file has the same name as the module in which it is included with a ".h" extension. For example, the header file for the module "gui.c" is "gui.h". Each header file contains declarations for including the various Motif header files required to use the Motif and Xlib functions. A header file associated with a module also declares all the external functions created in other modules and called by the module. The function prototypes for the functions in a module are also declared in its header file.

In addition, two header files are declared, called "global.h" and "structures.h". The former contains all the global variables used by the software tool and the latter declares the various data structures and constants that are shared by all the program modules.

All the modules mentioned above are managed by a makefile called "Makefile". The various modules are compiled separately and then linked together.

3.2.2 The User Interface

Various Motif widgets were used to develop QNT. All the necessary function calls to display the user interface of QNT are were made from the C program module "main.c". These functions are in the module "gui.c". The initial user interface for QNT is shown in Figure 11.

The user interface of QNT is divided into four different regions. Each region along with all the elements present in that region are discussed below.

- Region 1 of QNT comprises a blank area in which the logo for QNT is displayed. On

top of this area, a drawing area called the canvas is popped up inside a dialog shell when the user selects the "New" or "Open ..." option from the "File" pull down menu or the "Expand" option from the toolbox. The canvas consists of a Motif widget of the class XmDrawingAreaWidgetClass. It is placed inside a Motif shell widget of the class XmDialogShellWidgetClass. The canvas is created as a child of the dialog shell widget with a default width of 840 pixels and a height of 614 pixels. Elementary queueing network objects such as queues, servers, outsides, paths, and labels can be drawn in the canvas. The file name is displayed as the title of the window in which the canvas is displayed.

- Region 2 is the window in which the current action selected is displayed. It is located at the bottom of the user interface and below Region 1. The action selected could be any one of those that can be selected either from the four pull down menus or from the toolbox. If no action has been selected, the current action selected is displayed as "None".
- Region 3 of the user interface consists of three pull down menus. The three pull down menus are labeled "File", "Edit", and "Help". The "File" pull down menu consists of seven options: "New", "Open ...", "Close", "Save", "Save As ...", "Print ...", and "Quit".

  The "New" option in the "File" pull down menu permits the user to create a new file. A new file with a default file name of "Untitled" is created, opened, and a canvas that is empty is displayed provided no drawing has been performed in the current session by the user. Otherwise, the user is prompted to save the current file(s) through a "Save File" dialog box. The "Save File" dialog box provides three options to the user in the form of push buttons: "OK", "No", and "Cancel". If the "OK" push button

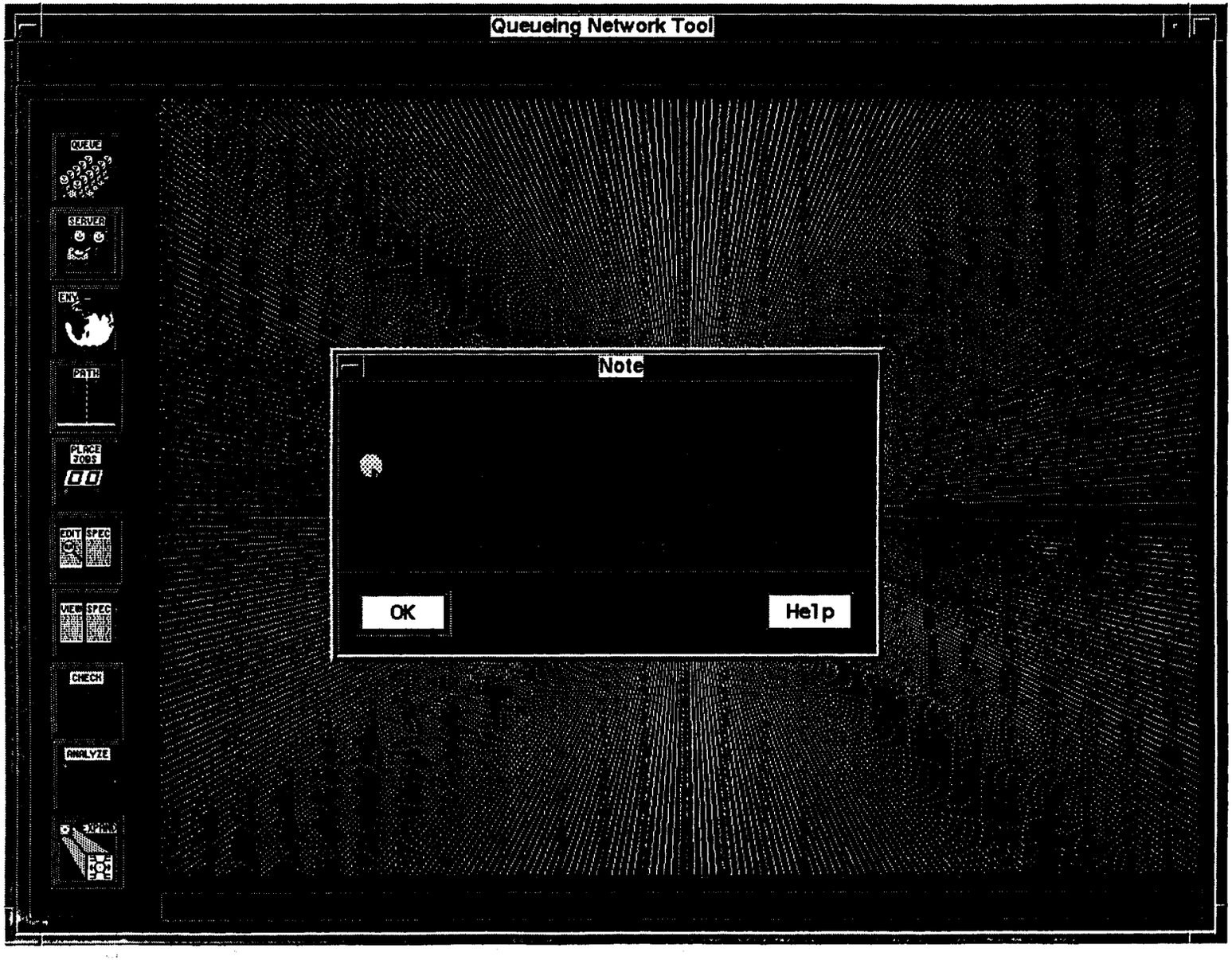

Figure 11. The four regions of QNT

is selected by the user, the current file is saved and a new file named "Untitled" is opened (see page 39 for detailed information on saving a file). If the "No" push button is selected by the user, the current file is not saved, the canvas is cleared, and a new file named "Untitled" is opened. Selecting the "Cancel" push button ensures that the "New" option chosen from the "File" pull down menu is cancelled.

The "Open ..." option in the "File" pull down menu permits the user to open an existing file. Selecting this option when no file is open displays a "File Selection" dialog box that contains two separate scrollable widgets. One of these widgets contains a list of directories while the other contains a list of files in the current directory. The current directory may be changed by using the "Filter" push button in the dialog box or by selecting the directory required from the scrollable Text widget that lists the directories. The filter option may also be used to display the list of files that satisfy a regular expression using wild cards (as in UNIX) in the scrollable Text widget that displays the file names. To change the current directory, the TextBox widget for the "Filter" should contain a valid path and the "Filter" push button in the dialog box should be selected. To select a file for opening, the file name can be typed in the TextBox widget for the "Selector", if the file is in the current directory. Otherwise, the full path starting from the root directory needs to be specified. A file can be opened either by selecting it from the scrollable List Box widget that displays the file names and clicking on the "OK" push button in the dialog box, by typing the file name in the TextBox widget for the "Selector" and clicking on the "OK" push button in the dialog box, or by double clicking on a file name in the scrollable ListBox widget. The file is then opened, the information in the file is read, and the

corresponding queueing network is displayed for the top-level queueing network provided the file is in the correct format. However, if the file selected is not in the correct format, a dialog box is displayed with an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. Then, the "Error Message" dialog box pops down and another file may be selected. When a file has been opened successfully, the "File Selection" dialog box is removed from the screen. Otherwise, it remains visible. The "Cancel" push button in the "File Selection" dialog box has been provided to pop down the "File Selection" dialog box without opening a file, thus effectively negating the "Open ..." option in the "File" pull down menu selected by the user. If there are file(s) open when the user selects the "Open ..." option from the file pull down menu, the user is prompted to save the current file(s) through a "Save File" dialog box. The "Save File" dialog box provides three options to the user in the form of push buttons: "OK", "No", and "Cancel". If the "OK" push button is selected by the user, the current file(s) are saved and the "File Selection" dialog box is displayed (see page 39 for detailed information on saving a file). If the "No" push button is selected by the user, the current file(s) are not saved, the canvas is cleared, and the "File Selection" dialog box is displayed. Selecting the "Cancel" push button ensures that the "Open ..." option chosen from the "File" pull down menu is cancelled.

The user can select the "Close" option in the "File" pull down menu to remove the active canvas. QNT then presents the user with a "Close" dialog box. The "Close" dialog box contains three push button: "OK", "No", and "Cancel". If the user selects the "OK" push button in the "Close" dialog box, QNT displays a "Save" dialog box to

the user if this layer of the queueing network has already been given a filename . The program then functions in a manner similar to that explained for the "Save" option in the "File" pull down menu (see page 39). If no file name has been provided, the "Save As ..." dialog box is presented to the user and QNT functions in a manner analogous to that explained for the "Save As ..." option in the "File" pull down menu. If the user selects the "Cancel" push button in the "Save" or "Save As ..." dialog box, the "Save" or "Save As ..." dialog box is popped down and control is returned to the "Close" dialog box. However, if the user selects the "OK" push button in the "Save" or "Save As ..." dialog box, both the "Save" dialog box or the "Save As ..." dialog box as well as the "Close" dialog box are popped down. If the user selects the "No" push button from the "Close" dialog box, the "Close" dialog box pops down and the active canvas is removed from the screen without saving any changes made to that canvas. If the user selects the "Cancel" push button in the "Close" dialog box, the dialog box pops down and the "Close" option in the "File" pull down menu selected by the user is cancelled.

To save the changes made to any layer of a queueing network, the user can select the "Save" option in the "File" pull down menu. This option does not close the file being saved. If an unnamed file (i.e., a queueing network with a file name "Untitled") is to be saved, a "Save" dialog box is displayed to the user that prompts for a file name. If the user selects the "OK" push button after entering the file name and no file name is entered by the user (i.e., a blank for a file name is considered invalid), or if the file name entered already exists for another file, an "Overwrite File" dialog box is displayed. It asks the user if he or she wants to overwrite the file. It provides two

push buttons: "OK" and "Cancel". If the user selects the "OK" push button in the "Overwrite File" dialog box, the "Save" and "Overwrite" dialog boxes are popped down and the file is overwritten with the data of the queueing network being saved. If the user selects the "Cancel" push button in the "Overwrite File" dialog box, it is popped down and the "Save" dialog box once again receives control to receive a new file name from the user. If the file name is valid, the file is written with the data of the queueing network being saved and the "Save" dialog box is popped down. The "Cancel" push button in the "Save" dialog box has been provided to pop down the "Save" dialog box and cancel the "Save" option in the "File" pull down menu selected by the user.

The "Save As ..." option in the "File" pull down menu permits the user to save the current version of the queueing network under a new file name while retaining the old version under the existing file name. Upon selecting this option, a "Save As" dialog box is displayed to the user. It contains a TextField widget wherein the current file name is displayed. The text displayed in this widget is in read-only mode (i.e., the file name cannot be edited). The dialog box also provides another TextField widget to enter the new file name. Upon entering the new file name and selecting the "OK" push button in the dialog box, the current queueing network will be saved under the new file name, the old file will be closed without saving the current changes, and the new file will be opened in the canvas. If the old file name is "Untitled", it implies that the old file was not named or saved. In this case the user is presented a "Save" dialog box instead of a "Save As" dialog box that prompts the user for a file name. Upon entering the file name and selecting the "Save" push button, the current drawing is saved under the new name. In both cases, the dialog boxes contain a "Cancel" push

button. When the user selects the "Cancel" push button, the "Save As ..." option in the "File" pull down menu selected by the user is cancelled. If the new file name provided by the user is the same as an existing file, an "Overwrite File" dialog box is displayed (see the explanation under the "Save" option from the "File" pull down menu on page 39 for details).

The user can select the "Print ..." option in the "File" pull down menu to print the QNT window and all drawing areas that are visible to a postscript file. When this option is selected by the user, QNT displays a "Print File" dialog box that prompts the user for the file name. If the user selects the "OK" push button after entering the file name and no file name is entered by the user (i.e., a blank for a file name is considered invalid), or the file name entered already exists for another file, an "Overwrite File" dialog box is displayed. It asks the user if he or she wants to overwrite the file. It provides two push buttons: "OK" and "Cancel". If the user selects the "OK" push button in the "Overwrite File" dialog box, the "Print File" and "Overwrite" dialog boxes are popped down and the file is overwritten with a postscript dump of the QNT window and the visible drawing areas in it. If the user selects the "Cancel" push button in the "Overwrite File" dialog box, it is popped down and the "Print File" dialog box once again receives control to receive a new file name from the user. If the file name is valid, the file is written with a postscript dump of the QNT window and the visible drawing areas in it and the "Print" dialog box is popped down. The "Cancel" push button in the "Print File" dialog box has been provided to pop down the dialog box and cancel the "Print ..." option from the "File" pull down menu selected by the user.

Selecting the "Quit" option in the "File" pull down menu terminates the application. Before exiting from the application, QNT presents the user with a "Quit" dialog box that prompts the user to save the current drawing. The "Quit" dialog box contains three push buttons: "OK", "No", and "Cancel". If the "OK" push button is selected by the user, the current drawing is saved, provided the drawing has already been named, and QNT terminates. If the drawing has not been named, the "Save" dialog box (see page 39) is displayed to the user. Note that all open canvases will be closed recursively. Hence, the push button selected applies to all the windows. If the "No" push button is selected by the user, QNT exits without saving any changes made to the current drawing. The "Cancel" push button ensures that the "Quit" option in the "File" pull down menu selected by the user is cancelled.

The "Edit" pull down menu provides three options, namely, "Undo", "Edit Label ...", and "Delete Element". The "Undo" option in the "Edit" pull down menu permits the user to reverse the last drawing action performed. This action can either be performed by selecting the "Undo" option from the "Edit" pull down menu or by clicking the third button of the mouse while it is located inside the canvas. As mentioned before, only drawing actions can be reversed by this option. Thus, the "Open ..." option in the "File" pull down menu is not undoable. For example, if an element in the queueing network has been deleted, selecting the "Undo" option would restore the element onto the canvas along with the data associated with that element. Performing an "Undo" operation once again would delete that element again.

The user can change the label of a node (i.e., a queue, a server, or an outside) by selecting the "Edit Label ..." option from the "Edit" pull down menu. This option

displays an "Edit Label" dialog box that prompts the user for the old label of the node as well as the new label. When this information has been entered and the "OK" push button has been clicked by the user, the software tool checks to ensure that the old label entered actually exists, is valid, and the new label does not match one of the existing labels. If the validity check succeeds, the old label is changed. Otherwise, an "Error Message" dialog box is displayed with an appropriate error message in it. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action.

To delete an element in the queueing network, the user would need to select the "Delete Element" option from the "Edit" pull down menu. The cursor would then change to an "X" while in the canvas. The user can now position the cursor over any element in the canvas and press the first button of the mouse to delete an element. The element could be a queue, server, outside, or a path. However, all paths that connect to or from the queue, server or outside that is to be deleted must first be deleted. If this is not done, an "Error Message" dialog box is displayed informing the user of this constraint. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action.

The "Help" pull down menu provides options, namely, "On Help ...", "Index ...", and "About QNT ...". The "On Help ..." option in the "Help" pull down menu displays an "On Help" dialog box providing the user with information on how to use the help system of QNT. It also provides an "OK" push button. When the user selects the "OK" push button, the "On Help" dialog box is popped down.

The "Help Index" dialog box is displayed when the user selects the "Index ..."

option from the "Help" pull down menu. It provides a scrolled list box in which the list of help topics are displayed. When the user clicks on one of the topics with the left mouse button, the information pertaining to it is displayed in the scrolled text widget in the dialog box. It also provides an "OK" push button. When the user selects the "OK" push button, the "Help Index" dialog box is popped down.

The "About QNT" dialog box is displayed when the user selects the "About QNT ..." option from the "Help" pull down menu. It displays the copyright information for QNT. It also provides an "OK" push button. When the user selects the "OK" push button, the "About QNT" dialog box is popped down.

- Region 4 of the user interface consists of a toolbox with ten action push buttons. It is located to the left of the canvas and below the pull down menus. The actions provided in the toolbox are used very frequently. The ten action push buttons are: "Queue", "Server", "Outside", "Path", "Place Jobs", "Edit Specification", "Show Specification", "Check", "Analyze", and "Expand".

The "Queue" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw a Queue". The user can now draw a queue represented by an open rectangle. The width of the open rectangle is fixed. When the user moves the mouse to the canvas and clicks the first button of the mouse, the current mouse coordinates are taken as the upper left corner of the queue. The user can now move the mouse to any location desired on the canvas, in effect stretching the open rectangle. During the stretching operation, the queue is shown as a dotted line. When the desired size of the queue has

been determined, the user can press the second button of the mouse to make the queue. The queue is provided with a default label that is displayed above the queue and the default parameters of the queue are set. To change the default label and parameters of the queue, the user can select the "Edit Label" option in the "Edit" pull down menu (see page 43) and the "Edit Specification" option in the tool box (see page 49), respectively.

The "Server" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw a Server". The user can now draw a server represented by a circle. The radius of the circle is fixed. When the user moves the mouse to the canvas and clicks the first button of the mouse, a dotted circle is made with the current mouse coordinates as the center of the server. Moving the mouse, moves the circle along with it, so that the server may be placed at any location desired in the canvas. When the user presses the second button of the mouse, the server is drawn permanently. The server is provided with a default label that is displayed above the server, and the default parameters for the server characteristics are set. To change the default label and parameters of the server, the user can select the "Edit Label" option in the "Edit" pull down menu (See page 43) and the "Edit Specification" option in the tool box (see page 49) respectively.

The "Outside" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw an Outside". The user can now draw an outside represented by a double line square with

an "E" inside it. The dimensions of the square are fixed. When the user moves the mouse to the canvas and clicks the first button of the mouse, a dotted double line square is made with the current mouse coordinates at the center of the outside. Moving the mouse moves the square along with it, so that the outside may be placed at any location desired in the canvas. When the user presses the second button of the mouse, the outside is drawn permanently. The outside is provided with a default label that is displayed above the outside and the default parameters for the outside characteristics are set. To change the default label and parameters of the server, the user can select the "Edit Label" option in the "Edit" pull down menu (see page 43) and the "Edit Specification" option in the tool box (see page 49) respectively.

The "Path" option may be selected by the user in the same manner as that described for the "Server" option. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw a Path". When the "Path" option is selected, a "Path Characteristics" dialog box is displayed. In the upper half of the dialog box, the user is prompted through two TextField widgets for the label of the source node and the label of the destination node between which the path is to established. The user is also prompted through a third TextField widget for the probability that a job moves along this path. If the user does not enter a value in this TextField widget, QNT assumes a default value of 1.0. The software tool checks for the validity of the labels entered once the user selects the "OK" push button in the dialog box. It does so by making sure that the source node label and the destination node label are valid, and a path does not already exist from the source node to the destination node. A valid label for the source node is in the set of existing queues,

servers, and outsides already drawn in the queueing network and a valid label for the destination node is not in the set of existing queues, servers, and outsides already drawn in the queueing network. In case the labels of the source node and/or destination node are invalid, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. The program also checks for the validity of the probability that a job moves along this path by ensuring that it is greater than zero and less than or equal to one. In case the validity check fails, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. If the path characteristics are valid, the dialog box pops down. The "Cancel" push button has been provide in the dialog box to cancel the "Path" option selected by the user. To draw a path, the user moves the mouse to the boundary of the source node and clicks the first mouse button. If the user clicks the mouse either inside or outside the boundary of a node, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message, while providing the source node and destination node labels for the path. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. When a valid source node has been selected, the mouse may be moved to the destination node. The path is made when the mouse hits the boundary of the destination node. The path between the source node and current mouse position is depicted by a dotted line. It is made permanent by drawing it as a

continuous line once the mouse hits the destination node. The direction of the path can be changed by the user by clicking the second button of the mouse and dragging the mouse. The path drawing action can be cancelled by the user at any time by clicking the second button of the mouse before the mouse touches the boundary of the destination node. Thereafter, the path may be deleted by selecting the "Undo" option in the "Edit" pull down menu or by clicking the third button of the mouse while it is in the canvas and before any other operation has been performed. If the user selects any other operation during the drawing of the path, the path drawing operation is automatically cancelled and no path is made. A path may have any number of segments and all the segments will have an arrow head attached to them.

The "Place Jobs" option, as the name suggests, permits the user to put jobs into a queue by pointing the mouse to the push button in the toolbox and clicking on it with the first button of the mouse. The user, upon selecting this option, is presented with a dialog box titled "Place Jobs". The current action displayed in Region 2 of the user interface is also updated to display "Place Jobs". The user is prompted for the label of the queue where the jobs are to be placed. The dialog box consists of four push buttons: "Increment", "Decrement", "OK", and "Cancel". Each time the user presses the "Increment" push button, the number of jobs in the designated queue is increased by one. Similarly, the "Decrement" push button reduces the number of jobs in the queue by one. When a queue is created, the number of jobs in the queue is set to zero. Upon clicking the "OK" push button in the dialog box, the software tool performs a number of validity checks. The primary checks include avoiding no label for the queue, a label for which no queue exists, a label that corresponds to a server

or an outside, and a negative number of jobs in the queue. In each case, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. If there are no errors, the "Place Jobs" dialog box is popped down and the number of jobs in the queue is updated.

The "Edit Specification" and "Show Specification" options provided in the toolbox perform similar functions. The specification for any node (queue, server, or outside) or path consists of the node or path label, the parameters specific to the node, the brief description of the node or path, and an interpretation for the functions represented by the node or path. The "Edit Specification" or the "View Specification" option is selected by the user by pointing the mouse to the push button in the tool box and clicking on it with the first button of the mouse. The current action displayed in Region 2 of the user interface is also updated to display "Edit a Specification" or "View a Specification". The "Edit Specification" option in the toolbox permits the user to enter and edit the parameters for a node or path, their brief description, and the interpretation for the node or path. On the other hand, if the "Show Specification" option has been selected by the user, the node or path label, its brief description, its interpretation, and the parameters of the node or path can be viewed only. When the user selects either of these options and the mouse is moved into the canvas, the cursor changes to a pencil. When the user selects a node or a path, the specification for that node is displayed in the "Edit Specification" or "Show Specification" dialog box. Depending on the option selected by the user, the information in the dialog box may

or may not be edited. The "Edit Specification" dialog box consists of two push buttons: "OK" and "Cancel" while the "Show Specification" dialog box contains only the "OK" push button. In either case, clicking the "OK" push button will save the specification for the node or path and pop down the dialog box. If the user selects the "Cancel" push button, the "Edit Specification" dialog box is removed from the screen without saving the current changes in the specification for the node.

The "Check" option in the toolbox may be selected by the user by pointing the mouse to the push button in the toolbox and clicking on it with the first button of the mouse. The current action selected is immediately updated in Region 2 of the user-interface to "Validate the Queueing Network". The application then checks whether each node satisfies the constraints of a queueing network. More specifically, it checks if for each node in the queueing network, the sum of the probabilities associated with each path to each node in its postset totals to 1.0. It also checks if there are any isolated nodes in the queueing network. If all nodes pass the validity test, it displays a "Validation Results" dialog box that informs the user that the queueing network being modeled is valid. Otherwise, is displays a dialog box with three scrolled Lists, one each for queues, servers, and outsides. Under each node type, it lists the labels of the nodes that did not pass the validity test. The dialog box also provides an "OK" push button for the user to pop down the dialog box if the user does not desire to correct the erroneous nodes or has already corrected them. If the user desires to correct an erroneous node, the user can double click on the label for that node in the List widget. This would display the "Postset Probability Correction" dialog box. This dialog box gives the label of the erroneous node and in a Scrolled Window the label of each node

in its postset and the probability associated with the path connecting the erroneous node to the node in the postset. The label of the node in the postset cannot be edited, but their associated probabilities can be edited by the user. The dialog box also provides an uneditable TextField widget where the total probability is shown. The user can move to the next TextField widget in the Scrolled Window by pressing the "Tab" key and can move to the previous TextField widget by pressing the "<Shift> Tab" key. Each time the user moves to the next or previous TextField widget, the total probability displayed in the dialog box is updated. The dialog box provides two push buttons: "OK" and "Cancel". If the user selects the "Cancel" push button, the changes made to the probabilities are not updated and the dialog box is popped down. If the user selects the "OK" push button, the changes are updated in the appropriate data structures and a validity check is once again performed on this node. If it now passes the validity test, its label is removed from the corresponding List widget in the "Validation Results" dialog box. Otherwise, it is retained in the List widget.

The "Analyze" option in the toolbox may be selected by the user by pointing the mouse to the push button in the toolbox and clicking on it with the first button of the mouse. The current action selected is immediately updated in Region 2 of the user interface to "Analyze the Performance of the Queueing Network". This option is used in order to analyze the queueing network as a whole, or only subnets of it. The application then displays the "Sub-graph Specification" dialog box that prompts the user for the label of the source node and destination node of the subnet that is to be analyzed. Since the current version of QNT is capable of analyzing only M/M/1 systems, QNT checks the validity of the subnet specifications entered by the user after

he or she has selected the "OK" push button in the "Sub-graph Specification" dialog box. A valid subnet is one in which there is a single direct path between the source and destination nodes and neither of these nodes have been expanded. The dialog box also provides a "Cancel" push button if the user decide not to go ahead with the analysis. In either case, the "Sub-graph Specification" dialog box is popped down. If the user selects the "OK" push button, QNT displays a "Sub-graph Characteristics" dialog box that displays the characteristics of the source and destination node in the subnet selected by the user as well as the characteristics of the path connecting them. The user can review these characteristics and edit them if desired. This dialog box also provides two push buttons: "OK" and "Cancel". If the user selects the "Cancel" push button, the changes made by the user are not updated, the dialog box is popped down, and the subnet is analyzed for its steady state performance characteristics. If the user selects the "OK" push button, the changes are updated in the relevant data structures (provided the changes are valid, otherwise appropriate error messages are displayed in dialog boxes), the dialog box is popped down, and the subnet is analyzed for its steady state performance characteristics. QNT then displays a "Performance Characteristics" dialog box that provides the user with characteristics of the different components of the subnet analyzed by the tool and its steady state performance characteristics. This dialog box provides four push buttons: "OK", "New", "Change ...", and "Dump ...". If the user selects any of the four push buttons, the dialog box is popped down. If the user selects the "OK" push button, no further analysis is performed. If the user selects the "New" push button, the "Sub-graph Specification" dialog box is displayed again for the user to enter the specifications of the new subnet

to be analyzed. If the user selects the "Change ..." push button, the "Sub-graph Characteristics" dialog box is displayed for the user to analyze the same subnet but with a different set of characteristics for the nodes and path in the subnet. If the user selects the "Dump ..." push button, the "Print" dialog box is displayed that functions in the same manner as that for the "Print ..." option in the "File" pull down menu (see page 41). The "Dump ..." push button creates an ASCII text file of the information displayed in the "Performance Characteristics" dialog box.

The "Expand" option in the toolbox may be selected to develop the next higher level of the queueing network. The "Expand" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. The current action selected is immediately updated in Region 2 of the user-interface to "Expand a Node in the Queueing Network". If the user now moves the mouse on the canvas, the mouse cursor changes to a hand lens. When the user selects a node for expansion, the software tool first checks to see if it is a valid node for expansion. A valid node is a queue, a server or an outside. If an invalid node is selected, no expansion is performed. If a valid node is selected, an "Expansion Filename" dialog box appears with a text widget for entering the name of the file, for the next level of expansion, for the node selected by the user and two push buttons: "OK" and "Cancel". If this the first time this node is being expanded, the text widget would be empty, otherwise, it would display the name of the expansion file already provided. In the former case, the user can enter a valid file name and select the "OK" push button. The software tool would then pop down the "Expansion Filename" dialog box and open a new canvas in a dialog shell on top of all the existing canvases. The

title of the dialog shell would give the name of the file, for this level of expansion, for the node selected by the user. The newly opened canvas would be empty if this is the first expansion at this level of the queueing network for the node being expanded. If the file name entered is not valid, an "Warning" dialog box is displayed that informs the user of the reason for the invalid file name. This dialog box contains an "OK" push button that would need to be selected by the user before any other action. When the user selects the "OK" push button in the "Warning" dialog box, it is popped down and control is returned to the "Expansion Filename" dialog box for entering a valid file name. If this is not the first time the node selected by the user is being expanded, the text widget in the "Expansion Filename" dialog box will contain the expansion file name previously provided. The user can choose to open this file by clicking on the "OK" push button. Alternately, the user can provide a new file name for the next level of hierarchy, for the node being expanded. The user can then click on the "OK" push button. If a new expansion file name is given by the user, a "Warning" dialog box is displayed that asks the user to confirm if he or she wants to change the expansion file name. This dialog box provides two push buttons: "OK" and "Cancel". If the user selects the "OK" push button in the "Warning" dialog box, this dialog box as well as the "Expansion Filename" dialog box are popped down and a new canvas is opened in a dialog shell on top of all the existing canvases. The title of the dialog shell would give the name of the file, for this level of expansion, for the node selected by the user. The newly opened canvas would be empty if the file name provided is a new one, otherwise it would contain any queueing network drawn at this level of the queueing network for the node being expanded. If the user did not provide a new file name but

desired to open the existing expansion file and selected the "OK" push button in the "Expansion Filename" dialog box, this dialog box would be popped down and a new canvas would be opened in a dialog shell on top of all the existing canvases. The title of the dialog shell would give the name of the file, for this level of expansion, for the node selected by the user. The newly opened canvas would contain any queueing network drawn at this level of the queueing network for the node being expanded. If the user selected the "Cancel" push button in the "Expansion Filename" dialog box, the dialog box is be popped down effectively negating the "Expand" option selected by the user from the toolbox. The user can make only one canvas active (completely visible) at one time. The user can do so by clicking on the canvas that needs to be made active. The QNT software tool would perform all operations (either from the toolbox or the menu) only on the active canvas except the "Quit" option in the "File" pull down menu. Selecting the "Quit" option in the "File" pull down menu would recursively close all the canvases and exit the application in a manner analogous to that explained under the "Quit" option. To close only the active canvas, the user may select the "Close" option in the "File" pull down menu (see page 39) or the "Close" option in the system menu of the dialog shell in which the canvas appears. There is a limit on the number of levels to which a node may be expanded. Since the canvases appear within dialog shells, they cannot be iconified separately. Further, they would be minimized if the application (QNT) is iconified.

3.2.3 Other Implementation Details

This subsection briefly discusses the implementation details associated with the

drawing of the arrow head for a path and some of the known limitations of QNT.

The drawing of the arrow head based on the direction of the path was a tricky implementation task. The code for drawing the arrow head was adapted from a similar application [Hassan93]. This section of the code is located in the "draw.c" module. The basic outline for drawing the arrow head is given below.

Let the end points for the path on which the arrow head needs to be drawn be (X_1, Y_1) and (X_2, Y_2). These are mouse coordinates in pixels. An arrow head is made of two lines A and B whose end points are (X_3, Y_3) and (X_4, Y_4), and (X_5, Y_5) and (X_6, Y_6), respectively. The coordinates of the two line segments A and B may be computed as given below.

$$X_3 = X_2$$

$$Y_3 = Y_2$$

$$X_4 = \text{Length} * \text{Cos} (\alpha - \beta)$$

$$Y_4 = \text{Length} * \text{Sin} (\alpha - \beta)$$

$$X_5 = X_2$$

$$Y_5 = Y_2$$

$$X_6 = \text{Length} * \text{Cos} (\alpha - \beta)$$

$$Y_6 = \text{Length} * \text{Sin} (\alpha - \beta)$$

where

Length = length of line A or line B

$$\alpha = \text{Tan}^{-1} ((Y_2 - Y_1) / (X_2 - X_1))$$

$$\beta = 20.0 * K$$

$$K = 0.01745329$$

The length of the line in pixels could either be fixed or proportional to the length of the path segment connecting the destination nodes. QNT uses the fixed version to draw the arrow heads. The length is set to be 15 pixels. The C language library function atan2 () is used to compute α. Line A and Line B are drawn using the Xlib function XDrawSegmets () to complete the arrow head drawing operation.

A limitation of QNT is that the maximum number of elements that can be drawn at any level of the queueing network is 2000. This limit is determined by the array size of the BUFFERELEMENT structure. The array size has been defined to be 2000 in the header file "global.h". This may be changed if desired. The default size of the canvas for any level of the queueing network has been set at 840 by 614 (width by height) pixels. This may also be changed if desired. Another limitation of QNT is that it is capable of analyzing only M/M/1 queueing systems, and cannot analyze subnets containing nodes which have been expanded. The individual canvases cannot be individually iconified. Further, multiple queueing network files cannot be opened at the same time if they are not related to one another in a hierarchical manner.

CHAPTER IV

EVALUATION OF THE TOOL

The testing and evaluation of the software tool, QNT, developed as part of this thesis are discussed in this chapter along with some comments on the drawbacks of this tool and suggestions for improvement based on the experience gathered in using the tool. About 45 students of the graduate level Operating Systems II class of Spring 1994 used QNT for two class assignments to design, develop, and analyze a timeshared system. It was observed that on the average 12 students used QNT simultaneously by running multiple copies of the tool. The evaluation of QNT was also performed by simultaneously running 10 copies of QNT to design and develop a queueing network model of an existing simulation package known as the Unified Simulation Environment (USE). The USE system is capable of prototyping and evaluating computer architectures and operating systems [Jhun92] [Hassan92] [Daily93] [Hassan94].

Useful feedback was received during the evaluation process that was used to improve the tool. The feedback indicated that there was a communication gap in the way the various options in QNT worked. Consequently, the help system of QNT was improved by providing detailed documentation on each of the options. Feedback also indicated that the "Save" and "Save As ..." options in the "File" pull down menu did not perform correctly when the full path was not specified along with the filename. This was corrected. A problem was also detected when QNT was run from a directory for which

the user had no write permission. This was corrected by creating the queueing network file in the users' home directory. As discussed in the next section, a number of smaller systems were also modeled using QNT.

4.1 Sample Systems Modeled by the Tool

As mentioned in the introduction to this chapter, one of the major systems modeled using QNT was the USE system. Figure 12 shows the queueing network model for a part of the USE system. The M/M/1 subnets of the system modeled were analyzed for their steady state performance characteristics. It was observed that the performance of the model was close to that expected from the actual system.

Other smaller systems were modeled using QNT and analyzed for their steady state performance characteristics. Figure 13 represents a closed queueing network model of a timeshared computer system (originally from [Lazowska84]) developed in a hierarchical manner. As is evident from the figure, the computer system modeled has three CPU's, S_1, S_2, and S_3. Jobs from the various users enter the queue Q_2 and after receiving service enter the queue Q_1 from where they go to the outside O_1 which represents the users of the computer system.

The closed queueing network shown in Figure 14 represents a software-level model (originally from [Lazowska84]). This model represents the steps involved in the development of a program using a high-level language.

It is to be noted that these models do not represent all the possible types of systems that can be modeled using queueing networks, or QNT for that matter, but are an indication of the range of systems that can be modeled.

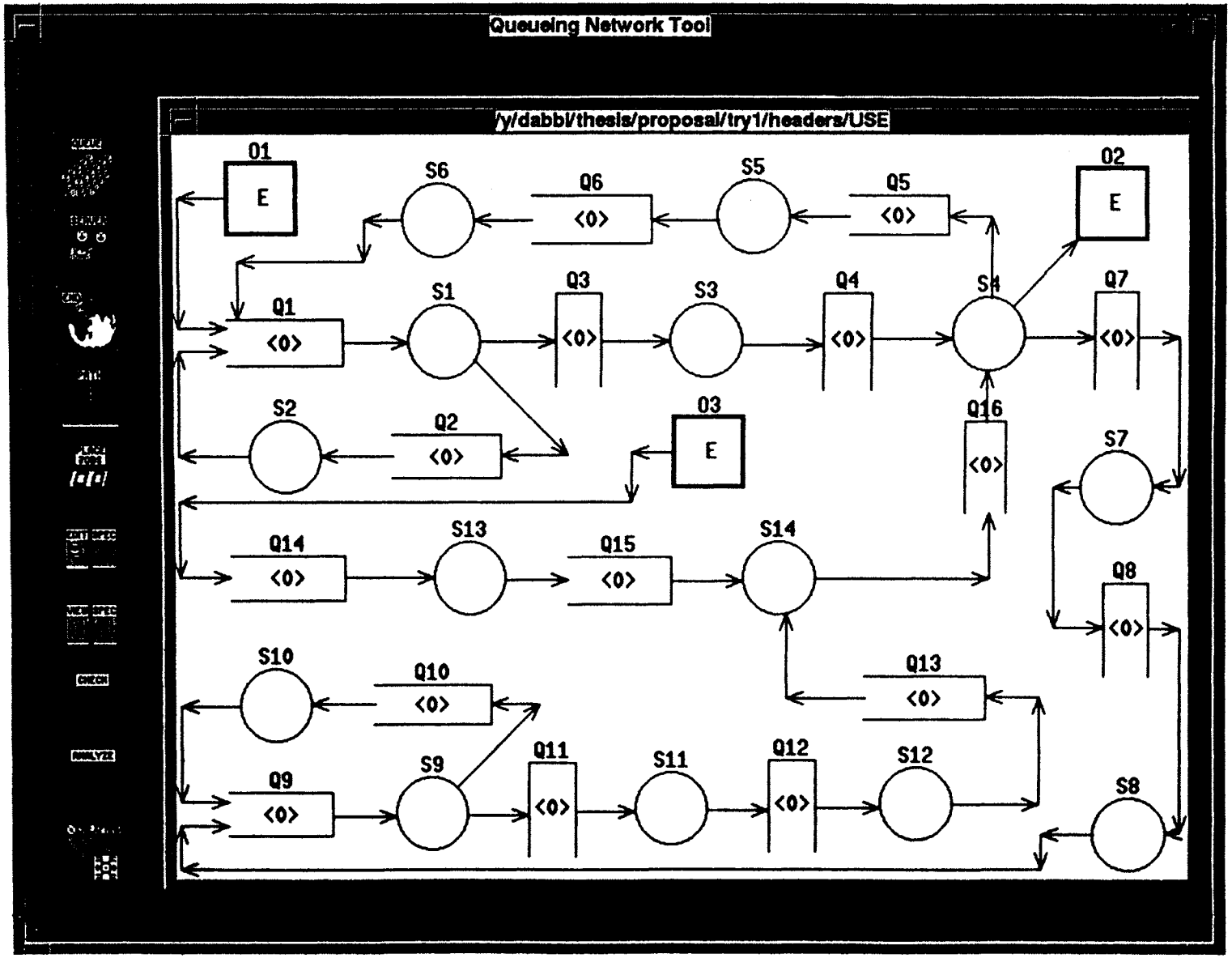

Figure 12. A queueing network model of the USE system

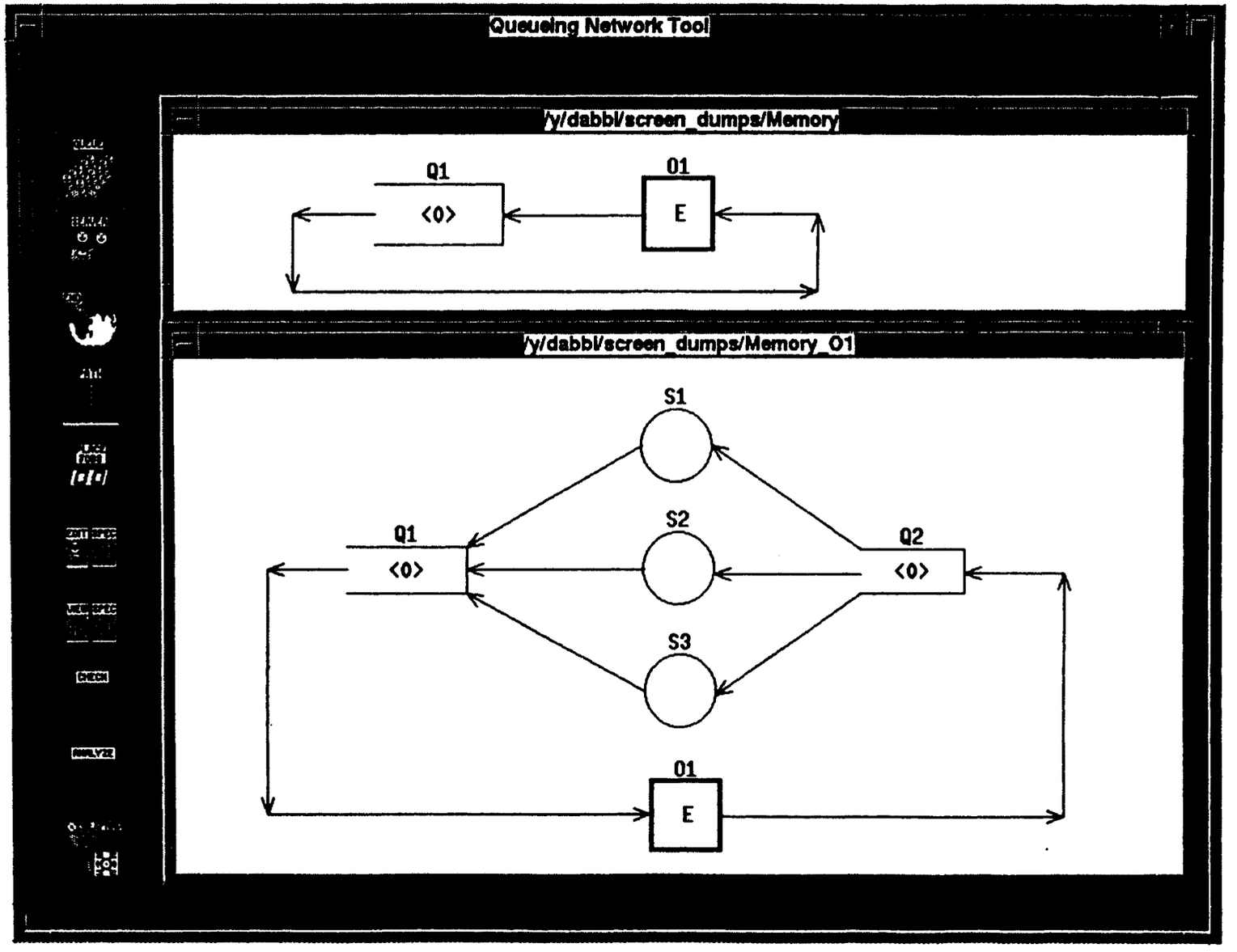

Figure 13. A hierarchical closed queueing network model of a timeshared computer system

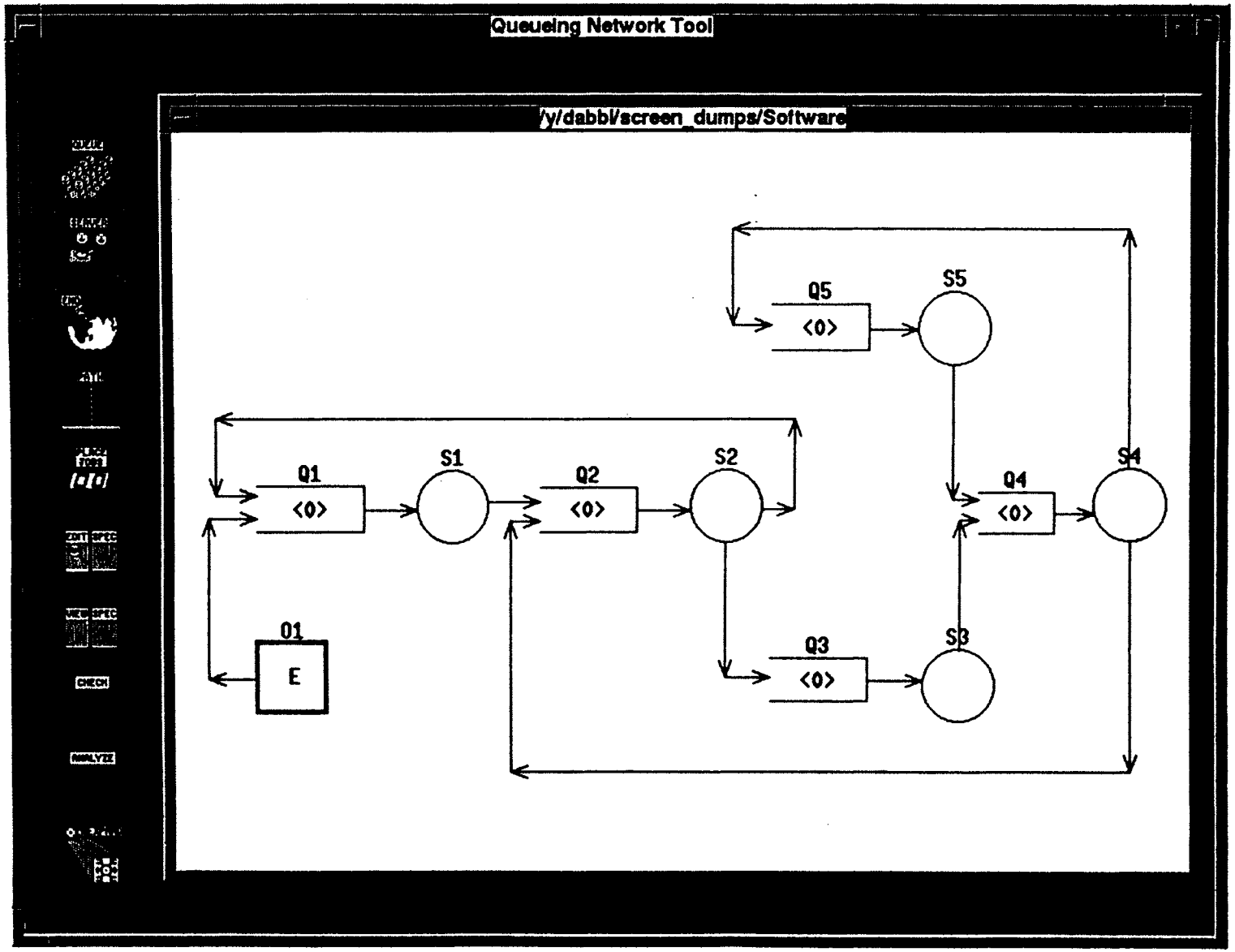

Figure 14. A software-level queueing network model

4.2 Observations

Based on the feedback received from the users of QNT (see Section 4.1 for details), several limitations of the software tool became apparent. Consequently, a number of improvements were incorporated into QNT. Two major improvements to the tool are described below.

- It was observed that one of the most heavily used option was the "Undo" option. Consequently, an attempt was made to minimize the number of actions required from the user to perform the "Undo" operation. Hence, in addition to selecting the "Undo" option from the pull down menu, a translation table was added to the software. The user can now perform the "Undo" operation with one mouse click by pressing and releasing the third mouse button with the pointer inside the canvas.
- A modification was made to the method of drawing a path between two nodes in a queueing network model. Initially, only single segment straight lines could be drawn between two nodes. This led to the problem of intersecting paths and paths that pass through nodes. Further, once a path was drawn there was no easy way to change its direction to avoid a node or an intersecting path. This resulted in a complicated and cluttered queueing network model that was unpleasant to view and difficult to understand. Hence, the path drawing algorithm was modified to enable the user to draw multiple segment paths between two nodes. There is no limit to the number of segments a path may have. To facilitate drawing multiple segment paths, the user can initiate the path drawing by pressing the first button of the mouse with it positioned inside the source node. The user can then move the mouse to any position on the

canvas and press the second button of the mouse to indicate the end of a segment. When the user points and clicks inside the destination node, the path is completed. In the case of a multiple segment path, the arrow will be made at the end of each segment.

CHAPTER V

SUMMARY AND FUTURE WORK

5.1 Summary

The importance of queueing networks as models of systems and the main objective of this thesis were discussed in Chapter I. Chapter II presented a survey of the current literature on queueing networks. The chapter provided an introduction to queueing networks by reviewing the fundamental definitions necessary to understand queueing networks. Chapter II also presented the characteristics of queueing networks; Erlang's model, Poisson distribution, and distribution of interarrival times; aggregation and branching of paths; analysis of an M/M/1 queueing network; modeling with queueing networks; and other related issues in queueing networks such as open networks of queues and closed networks of queues. The implementation details of the software tool developed, QNT (Queueing Network Tool), were discussed in Chapter III. The implementation platform and run-time environment including an introduction to the Sequent Symmetry S/81, the X Window System, and the OSF/Motif widget set were discussed in the earlier sections of Chapter III. Other sections in Chapter III considered the program structure of the software, the graphical user interface, and other implementation issues. Chapter IV provided the testing and evaluation methodology utilized for the software tool.

The main objective of this thesis was the development of a graphical software tool that can aid in the building of a queueing network model of a computer system and analyzing the steady state performance characteristics of the resulting queueing network or a sub-graph of the queueing network based on the movement of processes in the computer system. The current version of the tool is only capable of analyzing a queueing network or its sub-graphs that correspond to M/M/1 systems.

This tool can be used to design and construct systems of virtually any size and complexity. When the system to be designed is large and complex, it can be constructed using the tool in a hierarchical manner utilizing a top-down design approach. The graphical user interface was implemented using the OSF/Motif widget set (see Chapter III for a discussion of the X Window System and the Motif Toolkit). This tool was used to design computer systems of reasonable size and complexity in an academic environment. Based on the experience in using the tool and the difficulties encountered during its evaluation, steps were taken towards its improvement.

5.2 Future Work

The improvements mentioned below should be incorporated into the future versions of QNT.

- The cut, copy, and paste options can be implemented to provide additional flexibility in designing queueing networks and to reduce the planning required in drawing queueing networks using the tool. This would bring to the fore issues such as determining whether the subnet being cut or copied is a valid sub-graph.
- Options could be provided for the computation and analysis of various performance

measures of a queueing network or its sub-graph that are more complex than the simple M/M/1 system that the tool is currently capable of analyzing. It could also be capable of analyzing queueing networks that are designed in a hierarchical manner.

- The dialog shells, in which the canvases are displayed for drawing the queueing network in a hierarchical manner, could be provided a "Minimize" button so that they can be iconified separately. Further, QNT could be given the capability to access multiple queueing network files at the same time even if they are not related in a hierarchical manner.
- The software tool could be integrated into an existing simulation package called the USE system (the Unified Simulation Environment) designed for prototyping and evaluating computer architectures and operating systems [Jhun92] [Hassan92] [Daily93] [Hassan94], as was done in the case of DrawPetri (a graphical Petri Net modeling software tool) [Hassan93], not only to model different systems but also to use the resulting queueing network as an executable specification of the desired system.

REFERENCES

[Bakrabati91] N. Bakrabati, *X Window System Programming*, Macmillan Computer Publishing, Carmel, IN, 1991.

[Baskett75] F. Baskett, K. M. Chandy, R. R. Mains, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers", *Journal of the ACM*, vol. 22, no. 2, pp. 248-260, 1975.

[Berlage91] Thomas Berlage, *OSF/Motif: Concepts and Programming*, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Bruell80] S. C. Bruell and G. Balbo, *Computational Algorithms for Closed Queueing Networks*, North Holland Publishing, Inc., New York, NY, 1980.

[Buzen72] J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers", *Communications of the ACM*, vol. 16, no. 9, pp. 527-531, 1972.

[Byrant84] R. M. Byrant, A. E. Krzesinski, M. S. Lakshmi, and K. M. Chandy, "The MVA Priority Approximation", *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 335-359, 1984.

[Daily93] S.R. Daily and Mansur H. Samadzadeh, "Object-Oriented Simulation of Capability Based Architectures", *The Twenty-Sixth Annual Simulation Symposium*, Sponsored by SCS, IEEE-CS, and ACM, in conjunction with *The 1993 Simulation Multi-Conference*, pp. 258-266, Washington, DC, March 29-April 1, 1993.

[Fuller75] S. H. Fuller and F. Baskett, "An Analysis of Drum Storage Units", *Journal of the ACM*, vol. 22, no. 1, pp. 83-105, 1975.

[Gaver67] D. P. Gaver, "Probability Models for Multiprogramming Computer Systems", *Journal of the ACM*, vol. 14, no. 3, pp. 423-438, 1967.

[Gelenbe87] E. Gelenbe and G. Pujolle, *Introduction to Queueing Networks*, John Wiley and Sons Ltd., New York, NY, 1987.

[Gordon67] W. J. Gordon and G. F. Newell, "Closed Queueing Systems with

Exponential Servers", *Operations Research*, vol. 15, no. 2, pp. 254-265, 1967.

[Hassan92] Khaled M. Hassan and Mansur H. Samadzadeh, "An Object-Oriented Environment for Simulation and Evaluation of Architectures", *Proceedings of IEEE Twenty-Fifth Annual Simulation Symposium* in conjunction with *The 1992 SCS Simulation Multi-Conference*, Orlando, FL, pp. 91-97, April 1992.

[Hassan93] M.T. Hassan, "Toward a Graphical Petri Net Tool", Master of Science Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1993.

[Hassan94] K. Hassan and Mansur H. Samadzadeh, "Adding Virtual Memory to the USE Object-Oriented Simulation Environment", *Proceedings of the Object-Oriented Simulation Conference (OOS 1994)*, Simulation Series, Volume 26, Number 2, pp. 31-34, Tempe, Arizona, January 1994.

[Heidelberger82] P. Heidelberger and K. S. Trivedi, "Analytic Queueing Models for Parallel Processing with Asynchronous Tasks", *IEEE Transactions on Computer Systems*, vol. C-31, no. 11, pp. 1099-1108, 1982.

[Heidelberger83] P. Heidelberger and K. S. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency", *IEEE Transactions on Computer Systems*, vol. C-32, no. 1, pp. 73-82, 1983.

[Heller91] Dan Heller, *Motif Programming Manual*, O'Reilly and Associates, Inc., Sebastapol, CA, 1991.

[Hoyme68] P. K. Hoyme, S. C. Bruell, P. F. Afshari, and R. Y. Kain, "A Tree-Structured Mean Value Analysis Algorithm", *ACM Transactions on Computer Systems*, vol. 4, no. 2, pp. 178-185, 1968.

[Jackson57] J. R. Jackson, "Networks of Waiting Lines", *Operations Research*, vol. 5, no. 4, pp. 518-521, 1957.

[Jackson63] J. R. Jackson, "Jobshop Like Queueing Systems", *Management Science*, vol. 10, no. 1, pp. 131-12, 1963.

[Jhun92] Ik-Jeong Jhun, Khaled M. Hassan, and Mansur H. Samadzadeh, "Simulation of a Computing Environment Using Stochastic Processes and the Object-Oriented Technology", *Proceedings of the Twenty-Third Annual Pittsburgh Conference on Modeling and Simulation*, Vol. 22, Part 3, Edited By: William G. Vogt and Marlin H. Mickle, Pittsburgh, PA, pp. 1579-1585, April 30-May 1, 1992.

[Johnson90] E. F. Johnson and K. Richard, *Advanced X Window Application*

Programming, Advanced Computer Books, Management Information Source, Inc., Portland, OR, 1990.

[Keller90] B. J. Keller, *A Practical Guide to X Window Programming*, CRC Press, Inc., Boca Raton, FL, 1990.

[Kendall51] G. D. Kendall, "Some Problems in the Theory of Queues", *Journal of the Royal Statistical Society*, Series B, vol. 13, no. 2, pp. 151-185, 1951.

[Klienrock64] L. Klienrock, "Analysis of a Time-Shared Processor", *Naval Research Logistics Quarterly*, vol. 11, no. 1, pp. 59-73, 1964.

[Klienrock76] L. Klienrock, *Operating Systems: Computer Applications*, vol. 2, John Wiley and Sons Ltd., New York, NY, 1976.

[Lam83] S. S. Lam and Y. L. Lien, "A Tree Convolution Algorithm for the Solution of Queueing Networks", *Communications of the ACM*, vol. 26, no. 3, pp. 203-215, 1983.

[Lavenberg88] S. S. Lavenberg, "A Perspective on Queueing Models of Computer Performance", *CWI Monographs*, Edited by: O. J. Boxma and R. Syski, Elsevier Science Publishing Co., Inc., New York, NY, pp. 59-94, 1988.

[Lazowska84] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1984.

[Lewis71] P. A. W. Lewis and G. S. Shedler, "A Cyclic-Queue Model of System Overhead in Multiprogrammed Computer Systems", *Journal of the ACM*, vol. 18, no. 2, pp. 199-220, 1971.

[Lipsky77] L. Lipsky and J. D. Church, "Applications of a Queueing Network Model for a Computer System", *ACM Computing Surveys*, vol. 9, no. 3, pp. 205-221, 1977.

[Lipsky92] L. R. Lipsky, *Queueing Theory: A Linear Algebraic Approach*, Macmillan Publishing Co., New York, NY, 1992.

[Little61] J. D. C. Little, "A Proof of the Queueing Formula L = λW", *Operations Research*, vol. 9, no. 3, pp. 383-387, 1961.

[Maekawa87] M. Maekawa, A. E. Oldehoeft, and R. R. Oldehoeft, *Operating Systems: Advanced Concepts*, Benjamin Cummings Publishing Co., Menlo Park, CA, 1987.

[Mitra85] D. Mitra, "Probabilistic Models and Asymtotic Results for Concurrent Processing with Exclusive and Nonexclusive Locks", *SIAM Journal of Computing*, vol. 14, no. 4, pp. 1030-1051, 1985.

[Nutt92] G. J. Nutt, *Centralized and Distributed Operating Systems*, Prentice Hall, Inc., Englewood Cliff, NJ, 1992.

[Nye90] Andrian Nye, *X Protocol Reference Manual for Version 11 of the X Window System*, O'reilly and Associates, Inc., Sebastapol, CA, 1990.

[Scherr67] A. L. Scherr, *An Analysis of Time-Shared Computer Systems*, MIT Press, Cambridge, MA, 1967.

[Sequent90] *DYNIX/ptx User's Guide*, Sequent Computer, Inc., 1990.

[Thomasian86] A. Thomasian and P. F. Bay, "Analytic Queueing Network Models for Parallel Processing of Task Systems", *IEEE Transactions on Computer Systems*, vol. C-35, no. 12, pp. 1045-1054, 1986.

[Trivedi82] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1982.

[Wang85] Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems", *IEEE Transactions on Computer Systems*, vol. C-34, no. 3, pp. 204-217, 1985.

[Young90] D. A. Young, *The X Window System: Programming and Applications with Xt, OSF/Motif Edition*, Prentice Hall, Inc., Englewood Cliff, NJ, 1990.

[Zahorjan81] J. Zahorjan and E. Wong, "The Solution of Separable Queueing Network Models Using Mean Value Analysis", *ACM SIGMETRICS Performance Evaluation Review*, vol. 10, no. 3, pp. 80-85, 1981.

APPENDIXES

APPENDIX A

GLOSSARY AND TRADEMARK INFORMATION

accelerator: Single keystrokes that are the equivalent of certain application functionality, most commonly associated with menu selections.

aggregation of paths: The merging of two or more paths leading into a queue or a server.

application window: The window where an application resides with its complete user-interface.

arrival process: The distribution of arrivals into a queue either from within or from outside of a system.

background: The area on which a widget resides.

background color: The color from which all widgets generate their top and bottom shadows and their select color, and against which labels and bitmaps are created with the foreground color.

BCMP networks: (Baskett - Chandy - Mains - Palacios) A generalized class of mixed queueing networks that permits multiple job classes, allows open and closed subnets, varied queuing disciplines, and general service time distributions.

tripartite graph: A graph in which the set of vertices can be divided into three disjoint sets, and there exist no edges between any two vertices in the same set.

bitmap: An image created using only the foreground and background colors of the screen.

branching of paths: The division of a job stream emanating from a server, a queue, or an outside into two or more paths.

button: Either a physical button on the mouse or a widget that simulates a real button on the screen.

callback: A function or a procedure that is to be executed when a specific event occurs within a widget that is in a particular state.

class: A common description for a set of similar objects with the same structure but different attribute values. Each class has unique characteristics and any number of instances of the class may be created.

class hierarchy: A logical ordering of classes, in which each class lower in the hierarchy (sub-class) is a specialization of the class directly above it (super-class). Sub-classes may inherit, add, delete, or modify attributes.

click: Pressing and immediately releasing a mouse button without moving the mouse in between.

client: Any application that runs on the X Window System or any widget, or a set of widgets, that is managed to be displayed in a window on the screen.

client-server model: A server process in a client-server model provides some services to the other processes. These other processes are known as clients. In the X Window System, the server controls all input and output devices. An application is a client process that utilizes the services provided by the server.

closed queueing network: A queueing network that has a fixed number of jobs circulating in the system that neither enter or leave the network.

composite widget: A widget that contains one or more widgets as its children, and controls their geometry.

dialog box: A collection of widgets that are displayed by an application in response to an event when detailed information needs to be provided to the user or when input needs to be obtained from the user.

distribution of interarrival times: The probability distribution function of the time intervals between the arrival of two successive jobs in a queue either from another queue within the network or from outside the system.

ergodic system: A system in which the average values converge to those in the steady state.

Erlang model: A basic model for analyzing queueing networks. It suggests that any queueing model consists of a set of servers, a set of queues, and the directed paths between them.

event: A message from the X server to an application.

event handler: A procedure that is executed in response to one or more predefined events for a widget.

execution of a queueing network: The simulation of the dynamic movement of jobs in

a queueing network.

geometry management: The process of automatic negotiation of the size and relative position of all child widgets.

graphical user interface (GUI): A visual representation of some of the functionality of a computer that can be manipulated in a friendly, easy to use, and non-programmatic manner.

graphics context (GC): A data structure that contains various information necessary for drawing graphic objects on a window such as the foreground pixel, background pixel, line width, line style, and clipping region. A graphics context is applicable only to drawables that have the same depth and root window as the graphics context.

GUI: See graphical user interface.

icon: A graphical symbol of an object or an action. Selecting an icon typically results in either selecting the object or performing the action.

indegree: The number of paths terminating in a node on a directed graph.

inheritance: A mechanism that makes use of the characteristics of the super-class in a sub-class without the need for duplication.

inter-client communication conventions (ICCC): A set of protocols that govern the interaction among the clients as well as between a client and the window manager.

intrinsics: The base library of functions on which the Motif widget set has been built. It implements the fundamental procedures for building new widget classes.

jobs: Each queue is mapped to a non-negative integer i, which is interpreted as i jobs being present in that queue.

Kendall notation: A concise symbolism that describes the parameters of queues in a queueing network.

Little's law: A law that provides a relationship between the average queue length and the average waiting time for a job at the steady state.

load dependent systems: A system in which the mean service rate of a server fluctuates depending on the lengths of the queues being serviced by the server.

load independent systems: A system in which the mean service rate of a server is fixed and is independent of the lengths of the queues being serviced by the server.

Markov process: A sequence of states generated by a process that satisfies the Markov property which states that the probability that a system will be in a particular state at the next instant depends only on the current state.

mean queue length: The average number of jobs waiting for service including the job(s) being serviced.

mean turnaround time: The average length of time a job resides in the system to satisfy its service requirements.

mean value analysis: A technique for analyzing queueing networks that is based on expected values rather that stochastic distributions.

mean waiting line length: The average number of jobs waiting for service that are currently in a queue.

mean waiting time: The average time a job waits for having its service request satisfied.

memoryless: A property of a distribution that indicates no effect of history on the probability of occurrence of an event.

MULTIBUS: An industry standard for buses that may be used to connect a variety of peripheral devices.

normalization constant method: A technique for analyzing closed queueing networks that requires the evaluation of a normalization constant in exponential time.

open queueing network: A queueing network in which jobs enter from one or more points from outside a system and exit to one or more sinks outside the network.

outside: A node in a queueing network that is represented by a square to which jobs move to when leaving the system.

outdegree: The number of paths originating from a node in a directed graph.

paths: Directed arcs connecting queues, servers, and outsides that indicate the flow of jobs.

pixel: A single identifiable point on the screen or in a pixmap. A pixel may have different color values, or may be white or black in the case of a monochrome monitor.

pixel values: An n-bit value, where n is the number of bit planes in a window or a pixmap. In other words, n is the depth of the pixmap or window. In the case of a window, it indexes a colormap to derive the actual color to be made visible.

pixmap: A three-dimensional array of bits that can be considered as a two-dimensional array of pixels. The value of each pixel can range from 0 to 2^{n-1}, where n is the depth of the pixmap. Alternately, a pixmap may be viewed as a stack of n bitmaps.

pointer: A synonym for the mouse cursor.

Poisson distribution: A common statistical distribution for the number of jobs that arrive per unit time into a queue from either within the network or from outside the system.

processor utilization: The fraction of time a server is busy servicing jobs.

product-form matrix: A square matrix of size K (the number of queues in a system) in which each element of the matrix indicates the probability that a job in queue i after being served joins queue j.

protocol: A mutually agreed set of rules of cooperation between a client and a server in exchange for services from the server.

ready server: A server is said to be ready when one or more of the queues it is servicing has a job and there are no jobs currently being served by the server.

server (X Window System): It offers the basic windowing mechanism. It is responsible for handling inter-process communication connection between clients, graphic requests, and demultiplexes and screens. It is also responsible for multiplexing input back to the appropriate clients.

toolkit: A low-level library of objects and functions that are available for use to the application programmer and upon which the intrinsics are built.

tree convolution method: A technique used to a analyze closed queueing networks that is computationally more efficient in terms of time complexity and space complexity when compared to the normalization constant method and the mean value analysis technique.

widget: A user interface mechanism comprising data structures and the associated procedures that can be displayed in different ways such as menus, dialog boxes, or windows.

window: A rectangular area on the screen that belongs to a particular application.

window manager: The program that manages the display of windows and their manipulation on the screen.

X: A networked, portable, and transparent windowing system.

X client: An application program that makes use of the services of the X server for input and output.

Xm: The prefix for any value assigned to a widget resource. This convention differentiates the X Window System and Motif widget set values from values assigned to any other variables in the source code.

XmN: The prefix for any resource attribute whose value needs to be specified.

X protocol: The protocol by which X clients communicate with the X server.

X server: A set of C language routines that exclusively control the display hardware and service client requests.

Xt intrinsics: A synonym for X toolkit intrinsics.

X toolkit intrinsics: A library of functions, procedures, and data structures built on top of Xlib that makes application programming much easier compared to working with Xlib functions.

X Window System: A network transparent and hardware independent base layer that provides services to the graphical user interfaces.

TRADEMARK INFORMATION

APPENDIX B

USER GUIDE FOR QNT

1. Introduction

The queueing network modeling tool, QNT, is installed on the Sequent Symmetry S/81 computer system located at the Computer Science Department, Oklahoma State University, Stillwater, Oklahoma.

The QNT software tool provides a graphical environment to model a system using analytical techniques based on queueing theory. The OSF/Motif widget set was employed to develop the user interface for QNT. The startup screen for QNT provides different functions to the user through the three pull down menus and ten push buttons in the toolbox (see Figure 15). The title of the window in which the canvas appears indicates the name of the file being edited. The current action selected by the user is indicated in the window located at the bottom of the canvas. The queueing network of the system being modeled can be drawn on the canvas. To design and model complex systems, the queueing network can be drawn in a hierarchical manner using a top-down design approach. The following section describes all the available functions in QNT.

2. Using Menus

The various parts of the menu system in QNT consist of the menu bar, the menu pads, the menus, and the menu options. The purpose of the menu system is to allow the user to communicate with QNT. This section describes the various components of the menu system in QNT. The initial screen of QNT is shown in Figure 15.

Menu Bar: The *menu bar* is located in the upper left corner of the initial screen. It displays the titles for the various pull down menus.

Menu Pad: The *menu pad* consists of a title in the menu bar. The pull down menu associated with each menu pad can be displayed by the user by pointing the mouse to it and clicking the first mouse button.

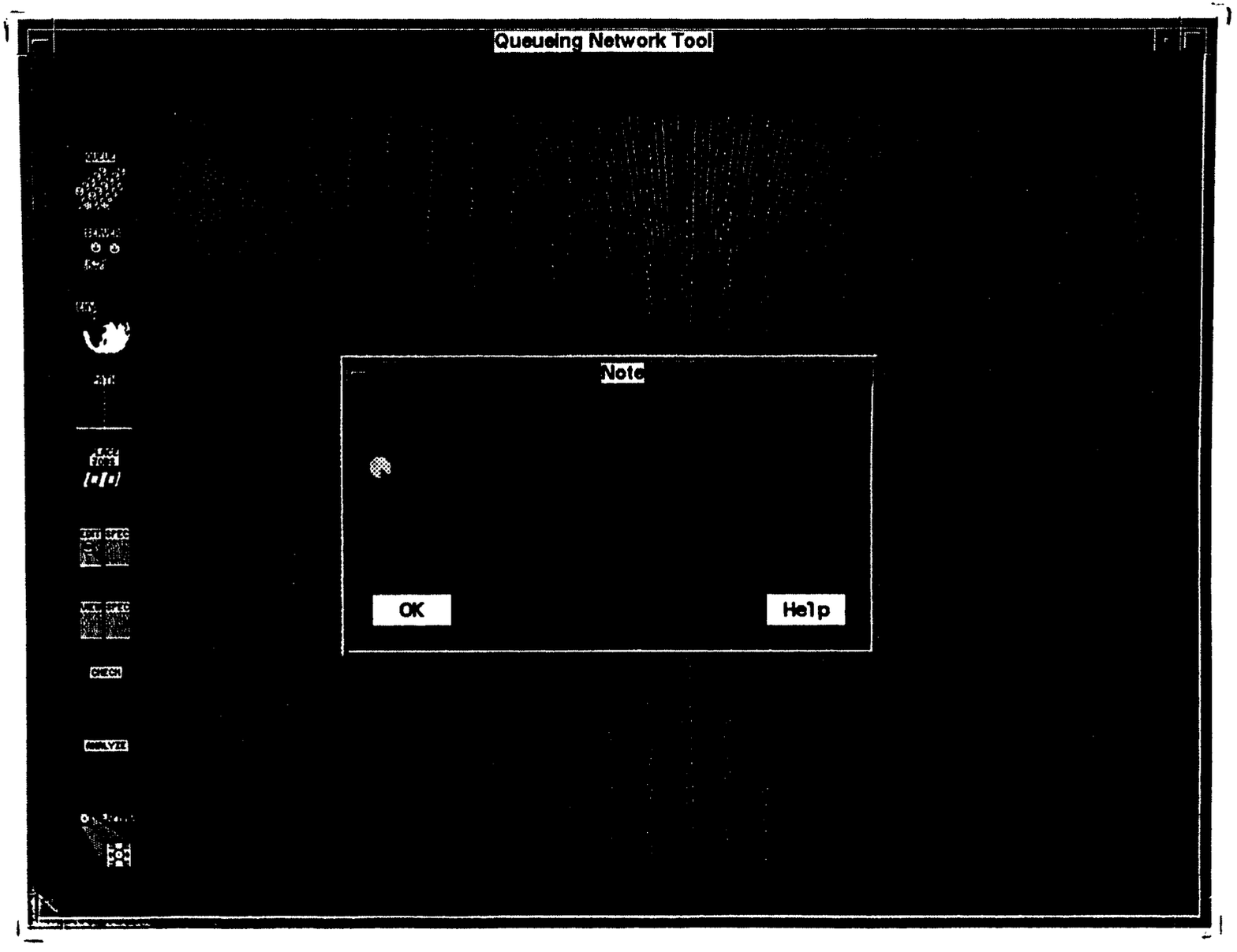

Figure 15. The initial screen of QNT

Menu: Each *menu* consists of a pull down menu pane that is displayed when the user points the mouse to the menu pad and clicks the first button of the mouse. A menu (also referred to as a pull down menu) is a list of options provided to the user. The user instructs QNT the action to be performed by choosing an option from the menu. The term "choosing" refers to the process of activating a selection (indicated visually by a highlighted option in the menu) by pointing and clicking the first button of the mouse.

Menu Option: As mentioned before, each menu consists of one or more options. Each of these options is known as a *menu option*. The menu options in each menu bear a logical relationship to the menu pad. The menu options within a menu that bear a close relationship to one another are grouped together and are separated by a horizontal line from the group or option immediately following it. The occurrence of an action in QNT is triggered by the selection of a menu option by the user. The action taking place may be the opening or closing of a window, the displaying of a dialog box, the execution of a command, or the selection of an action.

There are some menu options in QNT that are followed by an ellipsis (...). This indicates that additional information is needed from the user before the action pertaining to that menu option can be performed. A dialog box appears when such a menu option is selected by the user to request the additional information.

Some menu options are followed by a control-key combination. The user can use the control-key combination as a short cut for activating the corresponding menu option without displaying the menu and selecting the menu option.

To choose a menu option with a pointing device such as a mouse, point the mouse to the menu pad and hold the first button of the mouse down. The corresponding menu appears. With the first mouse button still pressed, drag the pointer to the menu option that is to be selected. Now release the first button of the mouse. The last menu option pointed to by the mouse, before releasing the first mouse button, is the one that is selected. The menu now disappears automatically.

3. Using Dialog Boxes

A user-interface object known as a dialog box appears when the user selects a menu option that is followed by an ellipsis. QNT also displays a dialog box at other times when it needs more information in response to an action chosen by the user. The user is required to provide additional information to the software tool through the dialog box and close it before taking any other action.

The user must exit the dialog box by clicking the first button of the mouse on either the "OK" push button or the "Cancel" push button of the dialog box before

taking any other action. The terminal would beep if the user attempts to perform any other action outside the dialog box such as clicking the mouse button in the canvas, on the menu bar, or in any of the push buttons in the toolbox. A dialog box is a composite widget that may consist of one or more child widgets such as push buttons, text boxes, and list boxes. These widgets are described below.

Push Button: A *push button* is a small rectangle that contains text describing the action performed by triggering the push button. The action associated with a push button occurs immediately after it is clicked with the first button of the mouse. This may also consist of displaying another dialog box.

Text Box: A *text box* is a rectangular box that is one or more lines in height and any width. It may be scrollable to display more text than can fit in the text box. It indicates an editable region where text may be entered. Click the first button of the mouse at any position of the text box to position the cursor at that location, then type and edit the text in the usual manner. There are descriptive text regions in QNT where the displayed text may or may not be browsed but cannot be edited, such as the window that displays the current action or the text box in the "Save As" dialog box that displays the current file name.

List Box: A *list box* consists of a list of items. The list of items could consist of the names of files, names of directories, etc. One or more items in the list may be selected by the user. To select an item in the list, point the mouse to the desired item and press the first button of the mouse. To select multiple items in the list, select the first item as mentioned above. The remaining items in the multiple selection list can be chosen by keeping the "Shift" key pressed, pointing the mouse to the item to be selected, and clicking the first button of the mouse. If the list is too long to fit in the list box, the user may browse through the list by pointing and clicking the first button of the mouse in the arrow at either end of the scroll bars.

Moving within a dialog box using the mouse: The user can move within the dialog box by moving the mouse to any control (push button or radio button), or any editable text box, and clicking the first button of the mouse.

Moving within a dialog box using the keyboard: The user can move within the dialog box using the keyboard only, from one control to another, by pressing the "Tab" key.

Repositioning a dialog box on the screen: It is possible to reposition the dialog box to any desired position on the screen provided it is completely within the main application window of QNT. To move the dialog box, move the mouse and point it to the title bar of the dialog box, press and hold down the first button of the mouse, and drag the mouse until the dialog box is in the desired location. The movement of the dialog box is indicated by its outline. Now release the mouse button.

4. File Menu

The "File" pull down menu provides the user with the functionality to open a new or an existing file, save a file, close a canvas, or exit from the application.

The "New" option in the "File" pull down menu permits the user to create a new file. A new file with a default file name of "Untitled" is created, opened, and a canvas that is empty is displayed provided no drawing has been performed in the current session by the user. Otherwise, the user is prompted to save the current file(s) through a "Save File" dialog box. The "Save File" dialog box provides three options to the user in the form of push buttons: "OK", "No", and "Cancel". If the "OK" push button is selected by the user, the current file is saved and a new file named "Untitled" is opened (see page 84 for detailed information on saving a file). If the "No" push button is selected by the user, the current file is not saved, the canvas is cleared, and a new file named "Untitled" is opened. Selecting the "Cancel" push button ensures that the "New" option chosen from the "File" pull down menu is cancelled.

The "Open ..." option in the "File" pull down menu permits the user to open an existing file. Selecting this option when no file is open displays a "File Selection" dialog box that contains two separate scrollable widgets. One of these widgets contains a list of directories while the other contains a list of files in the current directory. The current directory may be changed by using the "Filter" push button in the dialog box or by selecting the directory required from the scrollable Text widget that lists the directories. The filter option may also be used to display the list of files that satisfy a regular expression using wild cards (as in UNIX) in the scrollable Text widget that displays the file names. To change the current directory, the TextBox widget for the "Filter" should contain a valid path and the "Filter" push button in the dialog box should be selected. To select a file for opening, the file name can be typed in the TextBox widget for the "Selector", if the file is in the current directory. Otherwise, the full path starting from the root directory needs to be specified. A file can be opened either by selecting it from the scrollable List Box widget that displays the file names and clicking on the "OK" push button in the dialog box, by typing the file name in the TextBox widget for the "Selector" and clicking on the "OK" push button in the dialog box, or by double clicking on a file name in the scrollable ListBox widget. The file is then opened, the information in the file is read, and the corresponding queueing network is displayed for the top-level queueing network provided the file is in the correct format. However, if the file selected is not in the correct format, a dialog box is displayed with an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. Then, the "Error Message" dialog box pops down and another file may be selected. When a file has been opened successfully, the "File Selection" dialog box is removed from the screen. Otherwise, it remains visible. The "Cancel" push button in the "File Selection" dialog box has been provided to pop down the "File Selection" dialog box without opening a file, thus effectively negating the "Open ..." option in the "File" pull down menu selected by the user. If there are file(s)

open when the user selects the "Open ..." option from the file pull down menu, the user is prompted to save the current file(s) through a "Save File" dialog box. The "Save File" dialog box provides three options to the user in the form of push buttons: "OK", "No", and "Cancel". If the "OK" push button is selected by the user, the current file(s) are saved and the "File Selection" dialog box is displayed (see page 84 for detailed information on saving a file). If the "No" push button is selected by the user, the current file(s) are not saved, the canvas is cleared, and the "File Selection" dialog box is displayed. Selecting the "Cancel" push button ensures that the "Open ..." option chosen from the "File" pull down menu is cancelled.

The user can select the "Close" option in the "File" pull down menu to remove the active canvas. QNT then presents the user with a "Close" dialog box. The "Close" dialog box contains three push button: "OK", "No", and "Cancel". If the user selects the "OK" push button in the "Close" dialog box, QNT displays a "Save" dialog box to the user if this layer of the queueing network has already been given a filename . The program then functions in a manner similar to that explained for the "Save" option in the "File" pull down menu (see page 84). If no file name has been provided, the "Save As ..." dialog box is presented to the user and QNT functions in a manner analogous to that explained for the "Save As ..." option in the "File" pull down menu. If the user selects the "Cancel" push button in the "Save" or "Save As ..." dialog box, the "Save" or "Save As ..." dialog box is popped down and control is returned to the "Close" dialog box. However, if the user selects the "OK" push button in the "Save" or "Save As ..." dialog box, both the "Save" dialog box or the "Save As ..." dialog box as well as the "Close" dialog box are popped down. If the user selects the "No" push button from the "Close" dialog box, the "Close" dialog box pops down and the active canvas is removed from the screen without saving any changes made to that canvas. If the user selects the "Cancel" push button in the "Close" dialog box, the dialog box pops down and the "Close" option in the "File" pull down menu selected by the user is cancelled.

To save the changes made to any layer of a queueing network, the user can select the "Save" option in the "File" pull down menu. This option does not close the file being saved. If an unnamed file (i.e., a queueing network with a file name "Untitled") is to be saved, a "Save" dialog box is displayed to the user that prompts for a file name. If the user selects the "OK" push button after entering the file name and no file name is entered by the user (i.e., a blank for a file name is considered invalid), or if the file name entered already exists for another file, an "Overwrite File" dialog box is displayed. It asks the user if he or she wants to overwrite the file. It provides two push buttons: "OK" and "Cancel". If the user selects the "OK" push button in the "Overwrite File" dialog box, the "Save" and "Overwrite" dialog boxes are popped down and the file is overwritten with the data of the queueing network being saved. If the user selects the "Cancel" push button in the "Overwrite File" dialog box, it is popped down and the "Save" dialog box once again receives control to receive a new file name from the user. If the file name is valid, the file is written with the data of the queueing network being saved and the "Save" dialog box is popped down. The "Cancel" push button in the "Save" dialog box has been provided to pop down the "Save" dialog box

and cancel the "Save" option in the "File" pull down menu selected by the user.

The "Save As ..." option in the "File" pull down menu permits the user to save the current version of the queueing network under a new file name while retaining the old version under the existing file name. Upon selecting this option, a "Save As" dialog box is displayed to the user. It contains a TextField widget wherein the current file name is displayed. The text displayed in this widget is in read-only mode (i.e., the file name cannot be edited). The dialog box also provides another TextField widget to enter the new file name. Upon entering the new file name and selecting the "OK" push button in the dialog box, the current queueing network will be saved under the new file name, the old file will be closed without saving the current changes, and the new file will be opened in the canvas. If the old file name is "Untitled", it implies that the old file was not named or saved. In this case the user is presented a "Save" dialog box instead of a "Save As" dialog box that prompts the user for a file name. Upon entering the file name and selecting the "Save" push button, the current drawing is saved under the new name. In both cases, the dialog boxes contain a "Cancel" push button. When the user selects the "Cancel" push button, the "Save As ..." option in the "File" pull down menu selected by the user is cancelled. If the new file name provided by the user is the same as an existing file, an "Overwrite File" dialog box is displayed (see the explanation under the "Save" option from the "File" pull down menu on page 84 for details).

The user can select the "Print ..." option in the "File" pull down menu to print the QNT window and all drawing areas that are visible to a postscript file. When this option is selected by the user, QNT displays a "Print File" dialog box that prompts the user for the file name. If the user selects the "OK" push button after entering the file name and no file name is entered by the user (i.e., a blank for a file name is considered invalid), or the file name entered already exists for another file, an "Overwrite File" dialog box is displayed. It asks the user if he or she wants to overwrite the file. It provides two push buttons: "OK" and "Cancel". If the user selects the "OK" push button in the "Overwrite File" dialog box, the "Print File" and "Overwrite" dialog boxes are popped down and the file is overwritten with a postscript dump of the QNT window and the visible drawing areas in it. If the user selects the "Cancel" push button in the "Overwrite File" dialog box, it is popped down and the "Print File" dialog box once again receives control to receive a new file name from the user. If the file name is valid, the file is written with a postscript dump of the QNT window and the visible drawing areas in it and the "Print" dialog box is popped down. The "Cancel" push button in the "Print File" dialog box has been provided to pop down the dialog box and cancel the "Print ..." option from the "File" pull down menu selected by the user.

Selecting the "Quit" option in the "File" pull down menu terminates the application. Before exiting from the application, QNT presents the user with a "Quit" dialog box that prompts the user to save the current drawing. The "Quit" dialog box contains three push buttons: "OK", "No", and "Cancel". If the "OK" push button is selected by the user, the current drawing is saved, provided the drawing has already

been named, and QNT terminates. If the drawing has not been named, the "Save" dialog box (see page 84) is displayed to the user. Note that all open canvases will be closed recursively. Hence, the push button selected applies to all the windows. If the "No" push button is selected by the user, QNT exits without saving any changes made to the current drawing. The "Cancel" push button ensures that the "Quit" option in the "File" pull down menu selected by the user is cancelled.

5. Edit Menu

The "Undo" option in the "Edit" pull down menu permits the user to reverse the last drawing action performed. This action can either be performed by selecting the "Undo" option from the "Edit" pull down menu or by clicking the third button of the mouse while it is located inside the canvas. As mentioned before, only drawing actions can be reversed by this option. Thus, the "Open ..." option in the "File" pull down menu is not undoable. For example, if an element in the queueing network has been deleted, selecting the "Undo" option would restore the element onto the canvas along with the data associated with that element. Performing an "Undo" operation once again would delete that element again.

The user can change the label of a node (i.e., a queue, a server, or an outside) by selecting the "Edit Label ..." option from the "Edit" pull down menu. This option displays an "Edit Label" dialog box that prompts the user for the old label of the node as well as the new label. When this information has been entered and the "OK" push button has been clicked by the user, the software tool checks to ensure that the old label entered actually exists, is valid, and the new label does not match one of the existing labels. If the validity check succeeds, the old label is changed. Otherwise, an "Error Message" dialog box is displayed with an appropriate error message in it. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action.

To delete an element in the queueing network, the user would need to select the "Delete Element" option from the "Edit" pull down menu. The cursor would then change to an "**X**" while in the canvas. The user can now position the cursor over any element in the canvas and press the first button of the mouse to delete an element. The element could be a queue, server, outside, or a path. However, all paths that connect to or from the queue, server or outside that is to be deleted must first be deleted. If this is not done, an "Error Message" dialog box is displayed informing the user of this constraint. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action.

6. Help Menu

The "Help" pull down menu provides options, namely, "On Help ...", "Index ...", and "About QNT ...". The "On Help ..." option in the "Help" pull down menu displays an "On Help" dialog box providing the user with information on how to use the help system of QNT. It also provides an "OK" push button. When the user selects the "OK" push button, the "On Help" dialog box is popped down.

The "Help Index" dialog box is displayed when the user selects the "Index ..." option from the "Help" pull down menu. It provides a scrolled list box in which the list of help topics are displayed. When the user clicks on one of the topics with the left mouse button, the information pertaining to it is displayed in the scrolled text widget in the dialog box. It also provides an "OK" push button. When the user selects the "OK" push button, the "Help Index" dialog box is popped down.

The "About QNT" dialog box is displayed when the user selects the "About QNT ..." option from the "Help" pull down menu. It displays the copyright information for QNT. It also provides an "OK" push button. When the user selects the "OK" push button, the "About QNT" dialog box is popped down.

7. The Toolbox

The toolbox consists of ten push buttons. It is located to the left of the canvas and below the pull down menus. The actions provided in the toolbox are used very frequently. The ten action push buttons are: "Queue", "Server", "Outside", "Path", "Place Jobs", "Edit Specification", "Show Specification", "Check", "Analyze", and "Expand".

The "Queue" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw a Queue". The user can now draw a queue represented by an open rectangle. The width of the open rectangle is fixed. When the user moves the mouse to the canvas and clicks the first button of the mouse, the current mouse coordinates are taken as the upper left corner of the queue. The user can now move the mouse to any location desired on the canvas, in effect stretching the open rectangle. During the stretching operation, the queue is shown as a dotted line. When the desired size of the queue has been determined, the user can press the second button of the mouse to make the queue. The queue is provided with a default label that is displayed above the queue and the default parameters of the queue are set. To change the default label and parameters of the queue, the user can select the "Edit Label" option in the "Edit" pull down menu (see page 86) and the "Edit Specification" option in the tool box (see page 90), respectively.

The "Server" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw a Server". The user can now draw a server represented by a circle. The radius of the circle is fixed. When the user moves the mouse to the canvas and clicks the first button of the mouse, a dotted circle is made with the current mouse coordinates as the center of the server. Moving the mouse, moves the circle along with it, so that the server may be placed at any location desired in the canvas. When the user presses the second button of the mouse, the server is drawn permanently. The server is provided with a default label that is displayed above the server, and the default parameters for the server characteristics are set. To change the default label and parameters of the server, the user can select the "Edit Label" option in the "Edit" pull down menu (See page 86) and the "Edit Specification" option in the tool box (see page 90) respectively.

The "Outside" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw an Outside". The user can now draw an outside represented by a double line square with an "E" inside it. The dimensions of the square are fixed. When the user moves the mouse to the canvas and clicks the first button of the mouse, a dotted double line square is made with the current mouse coordinates at the center of the outside. Moving the mouse moves the square along with it, so that the outside may be placed at any location desired in the canvas. When the user presses the second button of the mouse, the outside is drawn permanently. The outside is provided with a default label that is displayed above the outside and the default parameters for the outside characteristics are set. To change the default label and parameters of the server, the user can select the "Edit Label" option in the "Edit" pull down menu (see page 86) and the "Edit Specification" option in the tool box (see page 90) respectively.

The "Path" option may be selected by the user in the same manner as that described for the "Server" option. Immediately, the current action displayed in Region 2 of the user interface is updated to display "Draw a Path". When the "Path" option is selected, a "Path Characteristics" dialog box is displayed. In the upper half of the dialog box, the user is prompted through two TextField widgets for the label of the source node and the label of the destination node between which the path is to established. The user is also prompted through a third TextField widget for the probability that a job moves along this path. If the user does not enter a value in this TextField widget, QNT assumes a default value of 1.0. The software tool checks for the validity of the labels entered once the user selects the "OK" push button in the dialog box. It does so by making sure that the source node label and the destination node label are valid, and a path does not already exist from the source node to the destination node. A valid label for the source node is in the set of existing queues, servers, and outsides already drawn in the queueing network and a valid label for the destination node is not in the set of existing queues, servers, and outsides already drawn in the queueing network. In case the labels of the source node and/or destination node are invalid, an "Error Message" dialog box is displayed that informs

the user of the error though an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. The program also checks for the validity of the probability that a job moves along this path by ensuring that it is greater than zero and less than or equal to one. In case the validity check fails, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. If the path characteristics are valid, the dialog box pops down. The "Cancel" push button has been provide in the dialog box to cancel the "Path" option selected by the user. To draw a path, the user moves the mouse to the boundary of the source node and clicks the first mouse button. If the user clicks the mouse either inside or outside the boundary of a node, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message, while providing the source node and destination node labels for the path. The user must respond to the error message in the dialog box by selecting the "OK" push button before taking any other action. When a valid source node has been selected, the mouse may be moved to the destination node. The path is made when the mouse hits the boundary of the destination node. The path between the source node and current mouse position is depicted by a dotted line. It is made permanent by drawing it as a continuous line once the mouse hits the destination node. The direction of the path can be changed by the user by clicking the second button of the mouse and dragging the mouse. The path drawing action can be cancelled by the user at any time by clicking the second button of the mouse before the mouse touches the boundary of the destination node. Thereafter, the path may be deleted by selecting the "Undo" option in the "Edit" pull down menu or by clicking the third button of the mouse while it is in the canvas and before any other operation has been performed. If the user selects any other operation during the drawing of the path, the path drawing operation is automatically cancelled and no path is made. A path may have any number of segments and all the segments will have an arrow head attached to them.

The "Place Jobs" option, as the name suggests, permits the user to put jobs into a queue by pointing the mouse to the push button in the toolbox and clicking on it with the first button of the mouse. The user, upon selecting this option, is presented with a dialog box titled "Place Jobs". The current action displayed in Region 2 of the user interface is also updated to display "Place Jobs". The user is prompted for the label of the queue where the jobs are to be placed. The dialog box consists of four push buttons: "Increment", "Decrement", "OK", and "Cancel". Each time the user presses the "Increment" push button, the number of jobs in the designated queue is increased by one. Similarly, the "Decrement" push button reduces the number of jobs in the queue by one. When a queue is created, the number of jobs in the queue is set to zero. Upon clicking the "OK" push button in the dialog box, the software tool performs a number of validity checks. The primary checks include avoiding no label for the queue, a label for which no queue exists, a label that corresponds to a server or an outside, and a negative number of jobs in the queue. In each case, an "Error Message" dialog box is displayed that informs the user of the error though an appropriate error message. The user must respond to the error message in the dialog

box by selecting the "OK" push button before taking any other action. If there are no errors, the "Place Jobs" dialog box is popped down and the number of jobs in the queue is updated.

The "Edit Specification" and "Show Specification" options provided in the toolbox perform similar functions. The specification for any node (queue, server, or outside) or path consists of the node or path label, the parameters specific to the node, the brief description of the node or path, and an interpretation for the functions represented by the node or path. The "Edit Specification" or the "View Specification" option is selected by the user by pointing the mouse to the push button in the tool box and clicking on it with the first button of the mouse. The current action displayed in Region 2 of the user interface is also updated to display "Edit a Specification" or "View a Specification". The "Edit Specification" option in the toolbox permits the user to enter and edit the parameters for a node or path, their brief description, and the interpretation for the node or path. On the other hand, if the "Show Specification" option has been selected by the user, the node or path label, its brief description, its interpretation, and the parameters of the node or path can be viewed only. When the user selects either of these options and the mouse is moved into the canvas, the cursor changes to a pencil. When the user selects a node or a path, the specification for that node is displayed in the "Edit Specification" or "Show Specification" dialog box. Depending on the option selected by the user, the information in the dialog box may or may not be edited. The "Edit Specification" dialog box consists of two push buttons: "OK" and "Cancel" while the "Show Specification" dialog box contains only the "OK" push button. In either case, clicking the "OK" push button will save the specification for the node or path and pop down the dialog box. If the user selects the "Cancel" push button, the "Edit Specification" dialog box is removed from the screen without saving the current changes in the specification for the node.

The "Check" option in the toolbox may be selected by the user by pointing the mouse to the push button in the toolbox and clicking on it with the first button of the mouse. The current action selected is immediately updated in Region 2 of the user-interface to "Validate the Queueing Network". The application then checks whether each node satisfies the constraints of a queueing network. More specifically, it checks if for each node in the queueing network, the sum of the probabilities associated with each path to each node in its postset totals to 1.0. It also checks if there are any isolated nodes in the queueing network. If all nodes pass the validity test, it displays a "Validation Results" dialog box that informs the user that the queueing network being modeled is valid. Otherwise, is displays a dialog box with three scrolled Lists, one each for queues, servers, and outsides. Under each node type, it lists the labels of the nodes that did not pass the validity test. The dialog box also provides an "OK" push button for the user to pop down the dialog box if the user does not desire to correct the erroneous nodes or has already corrected them. If the user desires to correct an erroneous node, the user can double click on the label for that node in the List widget. This would display the "Postset Probability Correction" dialog box. This dialog box gives the label of the erroneous node and in a Scrolled Window the label of each node in its postset and the probability associated with the path connecting the erroneous

node to the node in the postset. The label of the node in the postset cannot be edited, but their associated probabilities can be edited by the user. The dialog box also provides an uneditable TextField widget where the total probability is shown. The user can move to the next TextField widget in the Scrolled Window by pressing the "Tab" key and can move to the previous TextField widget by pressing the "<Shift> Tab" key. Each time the user moves to the next or previous TextField widget, the total probability displayed in the dialog box is updated. The dialog box provides two push buttons: "OK" and "Cancel". If the user selects the "Cancel" push button, the changes made to the probabilities are not updated and the dialog box is popped down. If the user selects the "OK" push button, the changes are updated in the appropriate data structures and a validity check is once again performed on this node. If it now passes the validity test, its label is removed from the corresponding List widget in the "Validation Results" dialog box. Otherwise, it is retained in the List widget.

The "Analyze" option in the toolbox may be selected by the user by pointing the mouse to the push button in the toolbox and clicking on it with the first button of the mouse. The current action selected is immediately updated in Region 2 of the user interface to "Analyze the Performance of the Queueing Network". This option is used in order to analyze the queueing network as a whole, or only subnets of it. The application then displays the "Sub-graph Specification" dialog box that prompts the user for the label of the source node and destination node of the subnet that is to be analyzed. Since the current version of QNT is capable of analyzing only M/M/1 systems, QNT checks the validity of the subnet specifications entered by the user after he or she has selected the "OK" push button in the "Sub-graph Specification" dialog box. A valid subnet is one in which there is a single direct path between the source and destination nodes and neither of these nodes have been expanded. The dialog box also provides a "Cancel" push button if the user decide not to go ahead with the analysis. In either case, the "Sub-graph Specification" dialog box is popped down. If the user selects the "OK" push button, QNT displays a "Sub-graph Characteristics" dialog box that displays the characteristics of the source and destination node in the subnet selected by the user as well as the characteristics of the path connecting them. The user can review these characteristics and edit them if desired. This dialog box also provides two push buttons: "OK" and "Cancel". If the user selects the "Cancel" push button, the changes made by the user are not updated, the dialog box is popped down, and the subnet is analyzed for its steady state performance characteristics. If the user selects the "OK" push button, the changes are updated in the relevant data structures (provided the changes are valid, otherwise appropriate error messages are displayed in dialog boxes), the dialog box is popped down, and the subnet is analyzed for its steady state performance characteristics. QNT then displays a "Performance Characteristics" dialog box that provides the user with characteristics of the different components of the subnet analyzed by the tool and its steady state performance characteristics. This dialog box provides four push buttons: "OK", "New", "Change ...", and "Dump ...". If the user selects any of the four push buttons, the dialog box is popped down. If the user selects the "OK" push button, no further analysis is performed. If the user selects the "New" push button, the "Sub-graph Specification" dialog box is displayed again for the user to enter the specifications of the new subnet

to be analyzed. If the user selects the "Change ..." push button, the "Sub-graph Characteristics" dialog box is displayed for the user to analyze the same subnet but with a different set of characteristics for the nodes and path in the subnet. If the user selects the "Dump ..." push button, the "Print" dialog box is displayed that functions in the same manner as that for the "Print ..." option in the "File" pull down menu (see page 85). The "Dump ..." push button creates an ASCII text file of the information displayed in the "Performance Characteristics" dialog box.

The "Expand" option in the toolbox may be selected to develop the next higher level of the queueing network. The "Expand" option is selected by the user by pointing the mouse to the push button and clicking on it with the first button of the mouse. The current action selected is immediately updated in Region 2 of the user-interface to "Expand a Node in the Queueing Network". If the user now moves the mouse on the canvas, the mouse cursor changes to a hand lens. When the user selects a node for expansion, the software tool first checks to see if it is a valid node for expansion. A valid node is a queue, a server or an outside. If an invalid node is selected, no expansion is performed. If a valid node is selected, an "Expansion Filename" dialog box appears with a text widget for entering the name of the file, for the next level of expansion, for the node selected by the user and two push buttons: "OK" and "Cancel". If this the first time this node is being expanded, the text widget would be empty, otherwise, it would display the name of the expansion file already provided. In the former case, the user can enter a valid file name and select the "OK" push button. The software tool would then pop down the "Expansion Filename" dialog box and open a new canvas in a dialog shell on top of all the existing canvases. The title of the dialog shell would give the name of the file, for this level of expansion, for the node selected by the user. The newly opened canvas would be empty if this is the first expansion at this level of the queueing network for the node being expanded. If the file name entered is not valid, an "Warning" dialog box is displayed that informs the user of the reason for the invalid file name. This dialog box contains an "OK" push button that would need to be selected by the user before any other action. When the user selects the "OK" push button in the "Warning" dialog box, it is popped down and control is returned to the "Expansion Filename" dialog box for entering a valid file name. If this is not the first time the node selected by the user is being expanded, the text widget in the "Expansion Filename" dialog box will contain the expansion file name previously provided. The user can choose to open this file by clicking on the "OK" push button. Alternately, the user can provide a new file name for the next level of hierarchy, for the node being expanded. The user can then click on the "OK" push button. If a new expansion file name is given by the user, a "Warning" dialog box is displayed that asks the user to confirm if he or she wants to change the expansion file name. This dialog box provides two push buttons: "OK" and "Cancel". If the user selects the "OK" push button in the "Warning" dialog box, this dialog box as well as the "Expansion Filename" dialog box are popped down and a new canvas is opened in a dialog shell on top of all the existing canvases. The title of the dialog shell would give the name of the file, for this level of expansion, for the node selected by the user. The newly opened canvas would be empty if the file name provided is a new one, otherwise it would contain any queueing network drawn at this level of the queueing

network for the node being expanded. If the user did not provide a new file name but desired to open the existing expansion file and selected the "OK" push button in the "Expansion Filename" dialog box, this dialog box would be popped down and a new canvas would be opened in a dialog shell on top of all the existing canvases. The title of the dialog shell would give the name of the file, for this level of expansion, for the node selected by the user. The newly opened canvas would contain any queueing network drawn at this level of the queueing network for the node being expanded. If the user selected the "Cancel" push button in the "Expansion Filename" dialog box, the dialog box is be popped down effectively negating the "Expand" option selected by the user from the toolbox. The user can make only one canvas active (completely visible) at one time. The user can do so by clicking on the canvas that needs to be made active. The QNT software tool would perform all operations (either from the toolbox or the menu) only on the active canvas except the "Quit" option in the "File" pull down menu. Selecting the "Quit" option in the "File" pull down menu would recursively close all the canvases and exit the application in a manner analogous to that explained under the "Quit" option. To close only the active canvas, the user may select the "Close" option in the "File" pull down menu (see page 85) or the "Close" option in the system menu of the dialog shell in which the canvas appears. There is a limit on the number of levels to which a node may be expanded. Since the canvases appear within dialog shells, they cannot be iconified separately. Further, they would be minimized if the application (QNT) is iconified.

APPENDIX C

SYSTEM ADMINISTRATOR GUIDE FOR QNT

1. Description

The QNT software is a graphical tool to model computer systems using analytical modeling based on queueing theory. To use QNT, the X Window System and the Motif widget set are necessary. The graphical user interface of QNT has been developed using the OSF/Motif widget set. The user interface of the tool consists of a menu, toolbox, windows for displaying messages and drawing the model along with dialog boxes, and other user-friendly features that make communication with the software easy for the user.

Although QNT has been designed for use with a mouse, the menu system, toolbox, and the various controls in the dialog boxes can also be traversed using the keyboard. However, from the standpoint of ease-of-use, it is recommended that a mouse be used. The startup screen of QNT consists of three pull down menus providing thirteen actions and a toolbox consisting of ten action push buttons. The current action selected is displayed in a separate window at the bottom of the user interface. To draw a queueing network model of a system, a canvas has been provided. To avoid the canvas from being cluttered, the model could be developed in a top-down hierarchical fashion. This could be done by selecting the expand option in the toolbox and clicking on a node in the queueing network. This would present a new canvas for drawing the next level of detail. The filename is displayed as the title of the canvas window.

2. Maintenance

The data structures used by the software tool for storing information about various objects drawn on the canvas are stored in the header file "structures.h".

The maximum number of nodes that any node can have in its preset is fifty. This limit has been defined in the variable MAXIMUMPRESETSIZE in the header file "structures.h". Similarly, the maximum number of nodes that any node can have in its postset is fifty, as defined in the variable MAXIMUMPOSTSETSIZE in the header file "structures.h". These limits may be changed as desired.

The maximum length of the message that describes the current action has been defined in the constant ACTIONMESSAGELENGTH and is set to one hundred characters. This constant has been defined in the header file "structures.h" that may be changed if desired.

The maximum length of any filename has been limited to fifty characters. This has been set in the variable FILENAMELENGTH in the header file "structures.h". It can also be modified.

The maximum length of the label of a node has been defined in the variable LABELLENGTH. This variable has been set to one hundred characters in the header file "structures.h", but may be changed when and if necessary.

The maximum length of the description for a node is limited to 300 characters. This has been defined in the variable SPECIFICATIONLENGTH in the header file "structures.h". This may also be modified as desired.

The information about the objects drawn on the canvas are stored in the data structure called GRAPHICELEMENTDATA. The maximum number of objects that may be stored in this data structure is five hundred. This limit has been set in the variable MAXIMUMOBJECTS in the header file "structures.h". This may be changed as desired.

The help messages and the error messages are located in the files "help.c" and "messages.c", respectively. To add additional help items and/or warning messages, these files need to be modified by adding a function for each additional item in the appropriate file(s).

To modify (add, change, or delete) the top-level menu system, the procedure "CreateMenuBar" in the file "gui.c" may be changed as needed. The function "GenerateMenuSystem" in file "gui.c" needs to be modified if the options in any of the pull down menus need to be changed. Further, the declarations for the menu items in the file "gui.c" also need to be modified.

The size of the main window for QNT is set to the maximum size of the screen. This size can be changed by modifying the XmNwidth and XmNheight resources of the form widget "Blackboard" to any desired size. It should be noted that the values of the XmNheight and XmNwidth resources are in pixels. The settings for these resources can be found in the file "main.c". Similarly, the size of the canvas can be changed by changing the XmNwidth and XmNheight resources of the drawing area widget "DrawingArea". The settings for these resources can be found in the file "expand.c".

The default values for the parameters of the queue, server, outside, and path are set when the dynamic memory for these data structures is allocated and the members of the data structure are initialized. These settings can be changed and can be found

in the file "allocate.c". The current default values are as follows:

Queue:	Arrival Rate	20 jobs per second
	Arrival Process	Erlang Distribution
	Queueing Discipline	FSCS
	Queue Capacity	Infinite
	Maximum Queue Capacity	Not Applicable
	Number of Jobs	0
Server:	Service Time	10000
	Service Process	Erlang Distribution
Outside:		There are no parameters for this node.
Path:	Probability	1.0

3. Options

QNT is a Motif widget set based software tool. As mentioned earlier, the Motif widget set is built upon the X Toolkit. Hence, QNT accepts all the standard X Toolkit command line options [Young90]. The more popular options are briefly described below.

-bg color	This option can be used to change the background color of an application window. The default background color is white.
-display display	This option is used to define the X server to which an application is to be connected.
-fg color	This option can be used to change the foreground color of an application window. The default foreground color is black.
-iconic	This option ensures that QNT is started by the window manager as an icon rather than as a normal window.
-rv	This option is used to swap the background and foreground colors of an application.
-title string	This option sets the title of an application window. This option may be ignored by the window manager. The default window title is the string specified after the -e command line option. If none is specified, the application name is used.

VITA

Jaganath Dabbi

Candidate for the Degree of

Master of Science

Thesis: TOWARDS A GRAPHICAL QUEUEING NETWORK TOOL

Major Field: Computer Science

Biographical:

Personal Data: Born in Bangalore, Karnataka, India, December 4, 1962, son of Mallikarjuna Rao Dabbi and Saraswathi Rao Dabbi.

Education: Graduated from Hyderabad Public School, Hyderabad, India, in April 1980; received Bachelor of Technology (Honors) Degree in Metallurgical Engineering from Indian Institute of Technology, Kharagpur, India, in May 1985; received Master of Business Administration Degree from Oklahoma State University, Stillwater, Oklahoma, in May 1993; completed requirements for the Master of Science Degree in Computer Science at Oklahoma State University in May 1994.

Professional Experience: Graduate Research Assistant, Office of Business and Economic Research, Oklahoma State University, May 1992 to June 1993; Graduate Instructor, Department of Management, Oklahoma State University, August 1991 to May 1992; Graduate Assistant, Department of Administrative Services, Oklahoma State University, August 1990 to May 1991; Executive Officer to C.E.O., The Tata Iron and Steel Company Limited, Jamshedpur, India, July 1985 to July 1990.