THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical Engineering
College of Engineering

Technical Report SEL-68-30

NETWORK MODELS FOR THE CONVERSATIONAL DESIGN OF STOCHASTIC SERVICE SYSTEMS

by

Victor L. Wallace
Keki B. Irani

November 1968

This research was supported by ARPA Contract DA-49-083-OSA-3050

ABSTRACT

Numerical analysis of Markovian queueing networks, and graphical communication of problem statements and results, offer a potential for truly conversational use of computers for "high-traffic" design of large-scale systems. However, one must develop a translator which converts the pictorial language of queueing network diagrams to the data structures required for efficient numerical analysis. This requires the development of new mathematical models for networks, and for the meaning of their components. Such mathematical models are explored, and their role in the development of adequate programming systems is described.

These models, algebraic in form, provide a vehicle for the information conveyed directly by the diagram, the information implicit in the symbology of the diagram, the information upon which the actual calculations are performed, and the procedure which transforms the information from the form of the diagram to the form for calculation.

TABLE OF CONTENTS

LIST OF FIGURES

NETWORK MODELS FOR THE CONVERSATIONAL DESIGN OF STOCHASTIC SERVICE SYSTEMS

V. L. Wallace
K. B. Irani

1. INTRODUCTION

It is frequently desirable to design stochastic service systems which cannot be adequately analyzed by normal queueing theoretic models. Such systems consist, in the most usual instances, of numerous waiting lines (or "queues"), servers, and controlling or directing stations which determine the discipline of task flow through the system. These systems are often realistic representations, for high-traffic design purposes, of behavior in diverse fields such as plant management, telephone switching, air traffic control, electronic warfare, logistics, and computer system specification and control.

Improved techniques for the design of such systems are greatly needed. For example, a design problem of this type which has not yet been adequately resolved is at the heart of a current crisis in the development of executive systems for demand-paged, multiprogrammed, timeshared computer systems. Problems of this type are also found frequently in the course of selecting the equipment and executive control strategies which permit a modern large-scale computer system to serve a specific environment.

These problems have, in the past, required painstaking study and could not be answered generally enough to be treated routinely. Increasingly, as the "computer utility" concept gains acceptance, these "queueing" problems will assume a more and more dominant position as the source of computer system inefficiency.

Similar needs for improved techniques are found everywhere that high-traffic problems occur. This is the natural result of the trend of every technology toward ever larger and more complex systems, with ever greater levels of traffic. New techniques should permit a more routine design of individual systems and strategies for specific environments.

One of the major hopes for significant improvement of design capabilities using queueing models lies in the so-called "conversational" computer techniques, whereby the calculating power of a computer can be closely coupled to the creative power of a design engineer. If a designer can freely pose alternative models to a computer and get immediate evaluation of various performance criteria, he may then generate enough insight (via cut-and-try procedures) to guess a near-optimal design for a system far too large or tightly interrelated to be treated by conventional optimization techniques.

However, the use of conversational techniques for the analysis and design of stochastic service systems requires development of a problem-oriented programming system which is specifically tailored to the demands of conversation. The input language must be terse, the calculations must be fast and reproducible, the variety of models available must be broad, and the results should be in a graphic, insight-provoking form.

The most promising approach to developing such a system is to use graphic displays for the man-computer communication, and to use recursive numerical methods [8] applied to Markovian models for the analysis. Graphical input to the computer in the form of queueing network diagrams provides an ideally compact medium for description of problems, while an output in the form of graphs provides a suitably insight-provoking form for results.

The recursive techniques provide fast calculations with excellent accuracy (hence reproducibility) of the calculated results [9]. They also can be applied to a wide variety of models. This variety of models is considerably broader than is feasible with closed form analysis, but not as general as the slower, less accurate simulation methods. The detail available in numerically solvable models is also midway between that available using closed form analysis and simulation methods.

There are three basic operations involved in a programming system to serve these goals. They are:

1) the servicing of the graphic operations,
2) the translation of the diagram to the form required for (Markov chain) input to the solution system,
3) the solution of the Markov chain.

The second of these is a process which, to our knowledge, has not been previously attempted, and has no obvious solution.

The subject of this paper is the exploration of mathematical models for Markovian queueing networks which can make this translation conceptually possible. (The models presented herein have been employed [3] in the development of specifications for such a translator, and an implementation of these specifications is currently in progress.)

In the next section we will take a closer look at the programming system as a whole in order to detail the role of the translator and its parts. Then, through the partial analogy of the better understood electric network models, Section 3 will attempt to provide a perspective for the queueing network models to be developed. The subsequent sections will develop various aspects of the models.

2. THE TRANSLATION PROCESS

Formal representations of Markovian queueing models in network diagram form are actually expressions in a graphical language. For example, the simple diagram of Figure 2.1 is a means of conveying a precise meaning

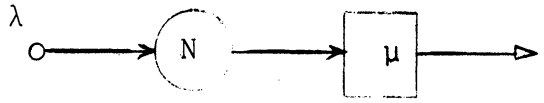

Figure 2.1

A Simple Network Diagram

(in the form of a Markov chain) through pictorial symbols and syntax. The small circle indicates a Poisson arrival source with mean intensity λ, the large circle indicates a queue with a maximum length of N, the rectangle indicates a server with mean service time $\frac{1}{\mu}$. and the triangle indicates an exit from the system. The lines indicate task flow according to rules established for each of the symbols joined.

The language of network diagrams is potentially a very rich one. It can contain symbols representing a considerable variety of sequencing rules, statistical properties, and selection processes. A further example, Figure 2.2, shows a network containing a random branch point, an overflow switch, and a merge point, in addition to the arrival source, queue, server, and exit.

The language, at present, must be confined to one expressing systems which define finite Markov chains with stationary transition probabilities, due to the

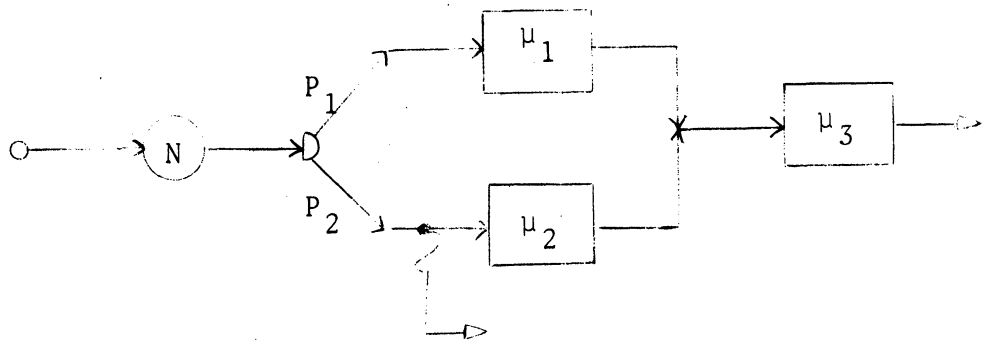

Figure 2.2

A Less Trivial Network Diagram

nature of the recursive numerical techniques to be used for solution. Convenience dictates further that the systems define continuous-time Markov chains. Specifications of the desired results are also limited to those results which can be obtained by some operation upon the equilibrium state probabilities. None of these restrictions are oppressive however, as much can be done within them [9].

To put the model requirements in clearer perspective, it is helpful to view the structure of the entire programming system in a little more detail. A schematic diagram

of such a system, as envisioned in this paper, is shown in Figure 2.3. (The diagram shows the various processors as oval symbols, and the information structures as rectangles.) The first thing to be noted is that, in a real graphics system involving cathode-ray displays, the input to the system is not actually a network diagram. Rather, it is a sequence of commands derived from the interpretation of light-pen or tablet stylus actions, and keyboard actions. These commands direct the construction of a diagram on the display screen and inform the programming system of the nature of the diagram and its desired output. The commands and the diagram are equivalent expressions in two graphic languages, often called a control language and a display language, respectively [7]. The control language is the actual form of the input to the translator.

A second observation about the system shown in Figure 2.3 is that the process of translating (indicated in the figure by the dotted rectangle) input commands from the control language to the form needed for analysis (the matrix structure) is divided into two stages, a command translator and a network translator. The intermediate form, which is a result of the command translator, is designated the network description structure.

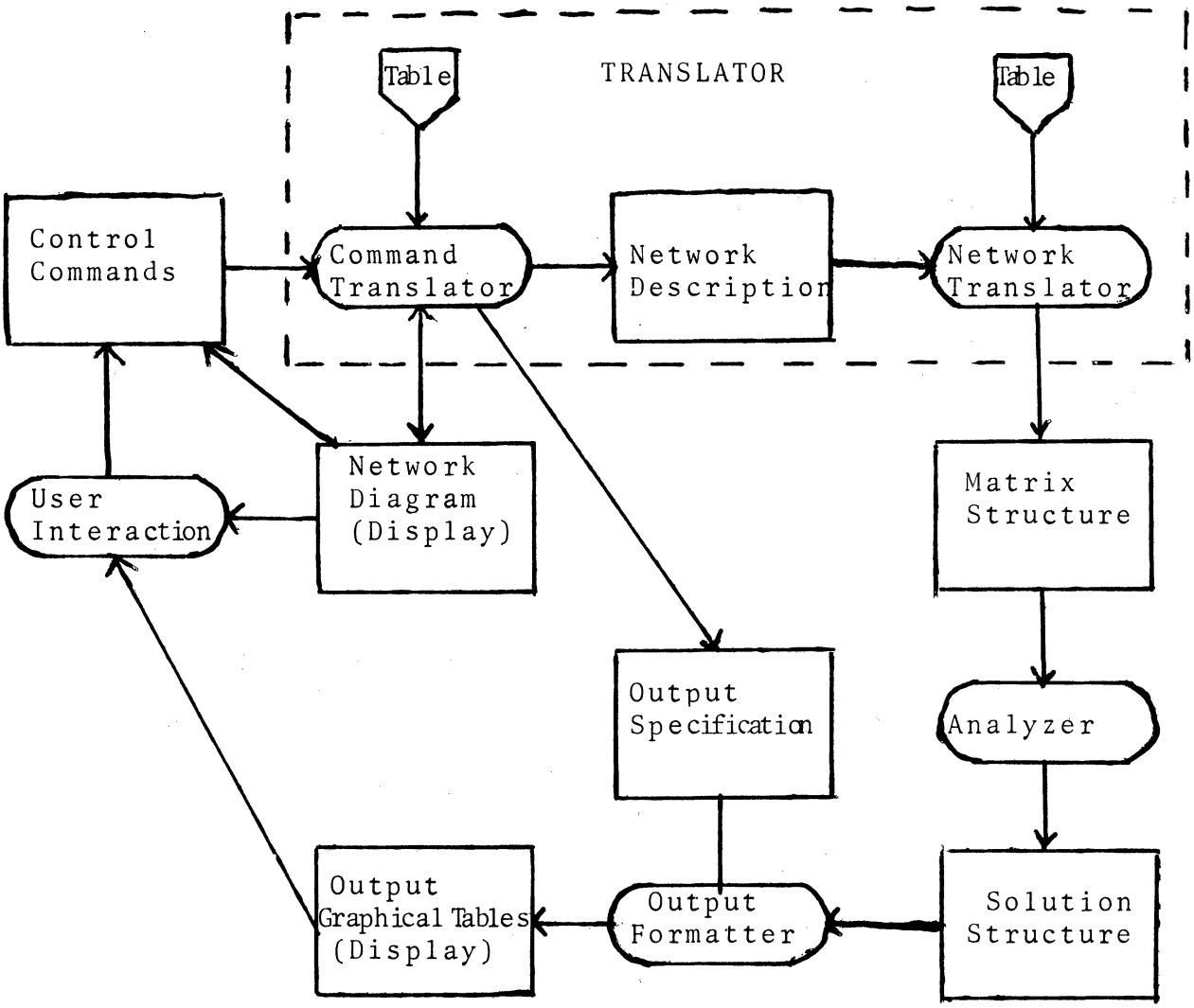

Figure 2.3

Functional Diagram of a Graphic, Conversational Programming System for Networks

The network description structure is the result of parsing the input expression and storing it in a form which can be more readily treated by a translator. Mnemonics and graphics which are irrelevant to the meaning of the input expression (where "meaning" here is the information needed to form the matrix structure representation) have been removed, and the result put in good systematic form. The practice of providing such an intermediate structure is well-established in programming-language translators because of simplifications it makes possible. In procedure-oriented compilers like MAD, ALGOL, or FORTRAN, the transformation to an intermediate structure is the first-pass operation which for algebraic statements often results in a list of triples, perhaps using Polish prefix form. (The form of this structure for queueing networks will be discussed in Sec. 4.)

Because of the potential richness of the network diagram language, and the interest in ease and flexibility of use, it is very desirable to encourage users to develop and use a shorthand notation. This is assumed to be accomplished by making the translators table-driven, and providing a facility for the user to define his own symbols and their meaning through special commands. The language treated is then an open-ended one, and the complex symbol graphics which would be needed to differen-

tiate among symbols of a large vocabulary can be avoided. Also, since few symbols are used in any single group of problems, some unnecessary memory requirements (both in the computer and in the user's brain) for seldom-used symbols are avoided.

The command translator associates with each symbol used in the commands a meaning supplied by the table. This meaning is both pictorial and symbolic, the pictorial meaning being used in the formation of the network diagram displayed, and the symbolic meaning being used in the formation of the network description structure. Some of the commands describe the solution desired and the form of its display. The command translator must also parse these and place them in a structure called the ouput specification structure. (The specifications so-described indicate, for example, which random variables of the queueing networks are to be examined, whether an expectation or a distribution function is to be displayed, and whether the result is to be displayed as a graph or a table.)

The network translator associates with each symbol used in the network description structure a meaning supplied by the associated table. This meaning is technical. For example, if a symbol is known to represent a "server," the network translator must replace the "server" symbol by technical information which tells us all we need to know

about how the "server-ness" of the component affects the qualities of the matrix structure. The network translator then can alter the form of this information to produce the matrix structure. The tables can be readily changed as the user's vocabulary changes.

The remainder of the system in Figure 2.3 is more or less self-explanatory. The analyzer operates upon the Markovian model described by the matrix structure and determines a vector of equilibrium state probabilities, which is the "solution structure" of the figure. These results are then processed according to the output specifications to provide the desired output of graphs and tables via the "output display file."

From this survey of the parts of the programming system, we have seen that the heart of the translator is a table-driven processor (the "network translator") which takes a parsed description (the "network description") of the symbolic network diagram, associates meaning with its symbols, and prepares an output (the "matrix structure") which is descriptive of a Markov chain in an easily solved form.

To develop a translator program, however, a mathematical model is needed which unequivocally answers the four basic questions:

(A) What is the useful information directly conveyed by the diagram (as abstracted for the "network description")?

(B) What is the information which must be presented to the analysis program (represented in the "matrix structure")?

(C) What is the algebraic procedure corresponding to the transformation from network diagram to "matrix structure?"

(D) What is the useful information indirectly conveyed by the symbols of the diagram (as represented in the "table" of the "network translator")?

These questions are not unusual in the context of programming languages. The first calls for syntactic description of the input language, the second calls for syntactic definition of the output language, the third calls for specification of the translation procedure, and the last calls for semantic definition of symbolic operators (like macro-definitions). However, models by which they can be answered for queueing networks have not been previously available.

The model will be algebraic in nature so that the translation procedure can be completely described in algebraic terms. Thus, it represents a new symbolic

framework in which queueing networks can be discussed with precision. Because of the novelty of the model, however, it is helpful in this report to precede its description by an analogous description of a model for electric networks, which are well understood. The functional diagram of Figure 2.3 is equally appropriate as a description of a system for analysis and design of electric networks, and therefore such a description will provide a good conceptual framework for all the work that follows.

3. ELECTRIC NETWORKS: A PARTIAL ANALOGY

It will be supposed, for simplicity, that the electric networks to bemodeled are linear, finite, passive, bilateral, and initially quiescent; that the solution required is a transient analysis of the voltage at particular nodes; and that sources will be solely current sources in the form of step functions. The network diagrams would be like that of Figure 3.1. These would somehow be translated into matrices which define sets of "node equations" for the networks.

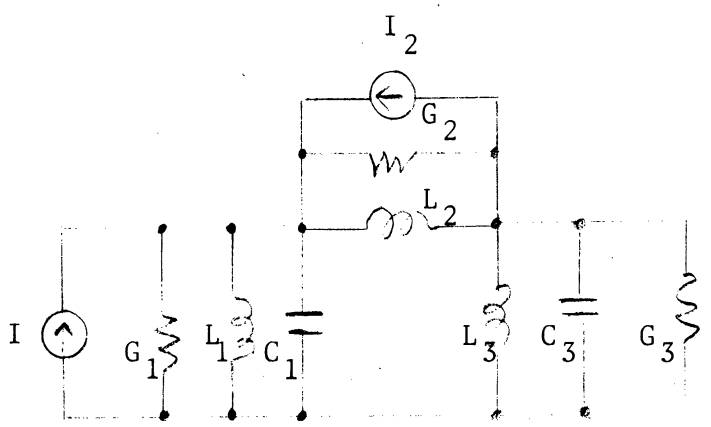

Figure 3.1

An Electric Network

An electric network diagram such as the one in Figure 3.1 consists of a collection of branches representing generic things like "resistors," "capacitors," and "current

sources." Each branch has two identifiable terminals, and the terminals of all the branches are joined in groups to form "nodes." In addition, each branch has a parameter (e.g., G_1, L_1, C_1, L_1, L_2, etc.) adjoined to it which modifies its generic description in some as yet undescribed manner.

We know, from classical electric network theory, that the meaning of this network diagram is ultimately a mathematical relationship between independent current source values and electrical potentials imagined to be present in the network. Making the usual assumption that the potentials are measured relative to the datum (bottom) node, there is one unknown potential (a function of time) for each nondatum node. If we represent these potentials by a vector V of their Laplace transforms, then the relationship can be expressed by the matrix equation

$$YV = I \qquad (3.1)$$

where Y is a matrix of transform admittances, and I is a vector of (Laplace) transformed node-driving currents. These latter two objects are uniquely determined from the information available in the diagram, and would represent the information stored in the matrix structure defined in

Figure 2.3. When they are known, the solution V can be obtained by well-known numerical methods.

Our purpose in this section is to show how Y and I can be routinely found from the network diagram, so that insight can be gained into the more complex procedure needed for queueing networks.

The first step is to transliterate the network diagram into a neater abstraction of its explicit content. Let a network N_e be defined simply by a collection L of branches and a collection K of nodes, so that

$$N_e = \langle L, K \rangle \tag{3.2}$$

$$L = \{\ell_j : j \in J_1\} \tag{3.3}$$

$$K = \{k_j : j \in J_2\}, \tag{3.4}$$

where J_1 and J_2 are index sets. Furthermore, let a branch ℓ_j, $j \in J_1$, be defined by the three things which identify it uniquely: a generic "type value" λ_j (which may take on values of "resistor," "capacitor," "inductor," and "current source"), a real parameter value σ_j (providing the value of resistance, capacitance, inductance, or current, as appropriate), and an ordered pair T_j of "terminals" (directivity is important to sources, and useful for the other branch types). Thus

$$\ell_j = \langle \lambda_j, \sigma_j, T_j \rangle, \quad j \in J_1. \tag{3.5}$$

A node k_j, $j \in J_2$, is merely a set of terminals such that

each terminal is contained by exactly one of the nodes in K . In other words, the k_j are defined so that

$$\bigcup_{j \in J_2} k_j = T \qquad (3.6)$$

and

$$k_{j_1} \cap k_{j_2} \in \emptyset, \text{ all } j_1, j_2 \in J_2, \qquad (3.7)$$

where $\emptyset$ is the empty set, and T is the set of all terminals in all the T_j, $j \in J_1$.

Thus, in this notation, an arbitrary network has been defined in terms of two primitive classes of objects: terminals, and generic types of branches. The first is merely a vehicle for topological (graph) information about the diagram. The second represents the generic identification of the graphics used for the branches in the diagram, and has the four possible values: "resistor," "inductor," "capacitor," and "current source," corresponding to the four graphics available to the drawer of the diagram. The model described by Eqs. (3.2) through (3.7) is an algebraic representation of the network description structure of Figure 2.3.

The second step is to extend this model of a network by replacing the branch-type designation by a model of its distinctive meaning. This meaning was implicit in the diagram, in the sense that the user left it understood in the notation of the diagram. The extension of the model

will make it explicit, and by so doing will allow us to use it in forming Y and I . If the model is general enough, it will also permit the introduction of new branch-type constants for the desirable kind of language flexibility proposed in the introduction.

A model to fit the purpose is readily found from electric network theory. The type designation of a branch ℓ_j can be replaced by a pair of quantities—the so-called short circuit current of the branch, and its Norton admittance—and satisfactorily fulfill the requirements. If we let i_j represent the Laplace transform of the former, and let y_j represent the latter, then the definition of the network N_e is extended by the equation

$$\lambda_j = \langle i_j, y_j \rangle, \quad j \in J_1 \,. \tag{3.8}$$

The two quantities are found for a given branch, when the type designator and parameter value are known, by inspection of Table 3.1. Recall that current sources were defined, for this system, to be strictly step functions, and the parameters of branches of current-source type were defined to be real constants, the amplitude of the step. Thus the transform of the short-circuit current of a source with parameter I_1 is I_1/s. The entries in the table represent the coefficients of a linear equation relating an assumed transformed branch current I_j to a

Table 3.1

Defining Branch Types

Type of Designator λ_j	Parameter Value σ_j	i_j	y_j
"Resistor"	G	0	G
"Inductor"	L	0	$\frac{1}{sL}$
"Capacitor"	C	0	sC
"Current Source"	I	$\frac{I}{s}$	0

transformed branch voltage V_j , in such a way that

$$I_j = i_j + y_j V_j \qquad (3.9)$$

Figure 3.2 defines this assumed pair of variables for a branch

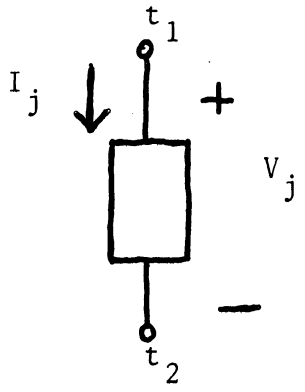

Figure 3.2

Defining Branch Voltage and Current

having terminal pair $T_j = \langle t_1, t_2 \rangle$. Table 3.1 plays the role of the "table" of the network translator in the programming system of Figure 2.3

Indeed, it is well known (by Norton's theorem) that any two-terminal fragment of the networks being treated here can be represented by a pair like that shown in Eq. 3.8. Thus, to define new fundamental symbols to represent any such two-terminal network fragment, one need only append additional lines to Table 3.1 If, in addition, vector value parameters (σ_j) are permitted, a model has been introduced which has considerable power to accept shorthand notation in the network diagrams. For example, if an appropriate new symbol were defined for the G-L-C parallel circuit, Figure 3.1 might have been drawn equivalently as Figure 3.3. The newly defined branches have a vector parameter giving conductance, inductance, and capacitance. Other ways to use the parameters, of course, also exist.

The third, and final, step in the process is to derive the matrix Y and the vector I from the network model described by Eqs. (3.2) - (3.8). For electric networks, this also is a simple operation, derived directly from Kirchoff's laws.

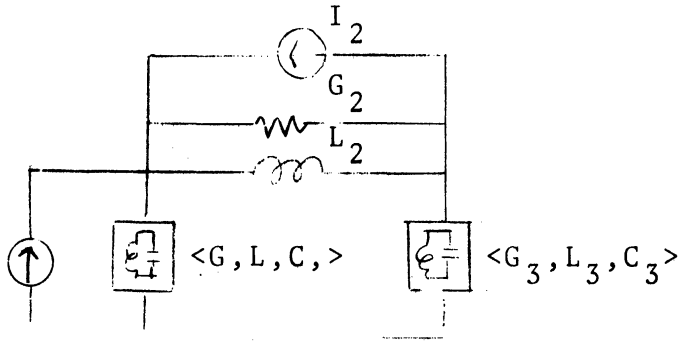

Figure 3.3

An Equivalent Network to Figure 3.1

Basic to the procedure is the calculation of a matrix A known as the branch-node incidence matrix. It can be routinely constructed from an inspection of the terminal pairs T_j, $j \in J_1$, and the terminal sets k_j, $j \in J_2$, called nodes. (The exact procedure is not important here.) Then, if we write the symbol i for a vector of the short circuit currents i_j for all branches, the node current vector I is simply

$$I = A^T i, \tag{3.10}$$

where A^T is the transpose of the matrix A. Furthermore, if we construct a diagonal matrix y whose diagonal elements are the Norton admittances y_j for each branch, then the matrix Y is simply

$$Y = A^T yA. \qquad (3.11)$$

This completes the description of the process of translating from diagram to matrix structure.

The four questions of Section 2 have been answered. Equations (3.2) through (3.7) represent the syntax of the network diagram, answering question A. The matrix Y and the vector I of Eq. (3.1) represent the object of translation, answering question B. The translation procedure is the described procedure for the creation of the matrix A, the calculation of I (via Eq. (3.10)), and the calculation of Y (via Eq. (3.11)), and is the answer to question C. Lastly, equation (3.8) and Table (3.1) represent the semantics of the elements, answering question D.

It is interesting to note that the discussion proceeded at at least three different levels of abstraction. First there was the network diagram, in which technological characteristics were understood but not in evidence. That is, things like voltages, currents, admittances, etc. were entirely suppressed. Second, there was the level of the models, in which all meaning was expressed in algebraic and functional terms. The operations were performed by

observing set memberships or by multiplying matrices and vectors. Third, there was the level of the technology represented by the models. At this level, the fundamental operations of the lower level are validated and explained in terms of physical laws (Ohm's law, Kirchoff's laws) and in terms of theorems about them (Norton's theorem, Superposition, etc.). These three levels will also be found in the development of the queueing network models.

4. DESCRIPTION OF QUEUEING NETWORK DIAGRAMS

In this section, the "syntax" of queueing network diagrams will be described as a means of answering the first of the four basic questions, "What is the information directly conveyed by the diagram?" The treatment will be analogous to that of electric networks (Sec. 3).

We postulate, from observation of queueing network diagrams like those of Figures 1.1 and 1.2, that a queueing network consists of a set of distinct objects which will be called elements, and a set of connections among them. The elements represent generic things like "queues," "servers," "sources," etc. They usually have parameters associated with them. The connections join distinct, identifiable parts of the element, which will be called ports. Only one connection can be made to a port.

There is some minor controversy over whether things like the random branch and the priority branch, as used in Figure 2.2, are rightly considered elements or connections. Philosophically, one can go either way. If they are elements, then connections are merely associations of pairs of ports. If they are connections, then every connection must be regarded as an instance of one of several types of connections (simple, random branch, priority branch, etc.), and it is sometimes necessary to associate parameters with them (viz., the branch

probabilities on the random branch). In this report, we have chosen the latter formulation (multiple connection types) because it results in simpler models of the elements when great generality of admissable element-types is not required. It is, thus, a useful compromise for a first attempt at implementation.

The essential information in the above description of the network diagram may be succinctly summarized by saying that a network N consists of a set E of elements and a set C of connections, and that an element e_i or a connection c_i consists of a type identifier τ_i, a parameter set ρ_i, and a port set P_i. Thus

$$N = \langle E, C \rangle \tag{4.1}$$

$$E = \{e_i : i \in I_1\} \tag{4.2}$$

$$C = \{c_i : i \in I_2\} \tag{4.3}$$

$$e_i = \langle \tau_i, \rho_i, P_i \rangle, \text{ all } i \in I_1 \tag{4.4}$$

$$c_i = \langle \tau_i, \rho_i, P_i \rangle, \text{ all } i \in I_2 \tag{4.5}$$

where I_1 and I_2 are disjoint index sets. Every port in the network belongs to exactly one element port set P_i, $i \in I_1$, and exactly one connection port set P_i, $i \in I_2$. The number of ports in a port set P_i is a function of the type and, in general, parameter values of the corresponding element or connection.

Equations (4.1) through (4.5) have an analogous purpose to that of Eqs. (3.2) through (3.7). They represent a mathematical model of queueing network diagrams. The element-types, connection-types, and ports are the primitive classes of objects in terms of which a class of things called queueing networks can be defined. This definition is a very general one, capable of allowing an enormous variety of networks to be represented. However, in particular instances of application it may be restricted, just as electric networks are often restricted to subclasses like "planar networks," "ladder networks," or the like. Furthermore, by restricting the allowed symbol types, the language can be further limited in specific applications, as in the case of "linear" electric networks, or "passive" electric networks.

5. A MODEL FOR THE MATRIX STRUCTURE

In this section we will concern ourselves with the second question describing models for the information represented in the "matrix structure," the output form of the translator.

The matrix structure must describe a finite state, continuous-time Markov chain $\{X_t: t \geq 0\}$ with stationary transition probabilities, in a form suitable for efficient numerical calculation of the equilibrium probabilities and efficient identification of the states to which the probabilities correspond. Traditionally, a Markov chain of the above type is described by a matrix $Q = (q_{r,s})$, $r,s \in S$, of transition intensities, where S is the set of states, and

$$q_{rs} = \frac{d}{dt} \; pr[X_t = s | X_0 = r]\Big|_{t=0^+}, \quad \text{all } r,s \in S, \tag{5.1}$$

$$q_{rs} \geq 0, \quad \text{all } r,s \in S;\ r \neq s \tag{5.2}$$

$$\sum_{s \in S} q_{rs} = 0, \quad \text{all } r \in S. \tag{5.3}$$

(An alternative interpretation of q_{rs} is that its reciprocal represents the conditional expectation of the time interval during which a sample $X_t(\omega)$ of the process remains in state r, given that it will jump from the state r to the state s at the end of the interval. It is also well known that, for these processes, all such intervals are exponentially distributed random

variables, independent conditionally upon the state r and s.)

The equilibrium probabilities π_r for each state r in S, are determined by a solution to the equations

$$\sum_{r \in S} \pi_r q_{rs} = 0, \text{ for } s \in S, \tag{5.4}$$

and

$$\sum_{r \in S} \pi_r = 1. \tag{5.5}$$

Thus, for our purposes, the information which must be provided as output from the translator (and hence input to the analyzer) must describe the state set S and the matrix Q.

The matrix Q is related to the network diagram in a complicated way, so that it is necessary to use an equivalent form which has an easy "physical" interpretation in terms of the network components, and yet can still be used like the matrix Q in numerical solution of Eq. 5.4. This equivalent form is developed by modeling the notion of "events" of the network.

For example, consider the network of Figure 2.1. In that example the states can be regarded as two-dimensional vectors $\langle x_1, x_2 \rangle$, the first coordinate (x_1) being the number of tasks in the queue, and the second (x_2) being the number of tasks in service (0 or 1). The state set is $S = \{\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 1,1 \rangle, \ldots, \langle N,1 \rangle\}$.

There are two "events" which occur: "task arrivals" and "task completions." If we can model these algebraically, and show that the model is equivalent to a description of Q, then we have a way of conceptually relating the diagram to the matrix. (In general, the events will be associated with specific network components in specific ways.)

One such model is to consider the "events" of a network as a set of triples whose first component describes the set of states at which the event can occur, whose second component describes the resulting state as a function of prior state, and whose third component describes the probability intensity of occurrence of the event. Proceeding with our example, the states for which an "arrival" can occur are all those for which the queue is not full, and it results in an increment of x_1 by one, except when the server is empty ($x_2 = 0$) when it increases x_2 by one. This event occurs with probability intensity λ, so that the event could be written

$$\xi_1 = \langle b_1, h_1, \mu_1 \rangle \tag{5.6}$$

where

$$b_1 = S - \{\langle N,1\rangle\} = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,1\rangle, \ldots, \langle (N-1),1\rangle\} ,$$

$$h_1(x_1,x_2) = \begin{cases} <0,1>, & \text{when } <x_1,x_2> = <0,0> \\ <x_1+1,x_2>, & \text{elsewhere in } b_1, \end{cases}$$

and

$$\mu_1 = \lambda.$$

Similarly, the "completion" event can occur only when $x_2 > 0$, it results in a decrease of queue length x_1 whenever $x_1 > 0$ or in emptying the server if $x_1 = 0$, and it occurs with probability intensity μ.

Thus, the event can be written

$$\xi_2 = <b_2,h_2,\mu_2> \tag{5.7}$$

where

$$b_2 = S-\{<0,0>\} = \{<0,1>,<1,1>,\ldots,<N,1>\}$$

$$h_2(x_1,x_2) = \begin{cases} <0,0>, & \text{when} <x_1,x_2> = <0,1> \\ <x_1-1,x_2>, & \text{elsewhere in } b_2, \end{cases}$$

and

$$\mu_2 = \mu .$$

In this example, then, the set of events is

$$\Xi = \{\xi_1,\xi_2\} . \tag{5.8}$$

Following this approach in general, let Ξ, the set of events of a network, be an indexed set

$$\Xi = \{\xi_k : k \in K_1\}, \tag{5.9}$$

where, for all $k \in K_1$, ξ_k is a triple

$$\xi_k = \langle b_k, h_k, \mu_k \rangle, \tag{5.10}$$

and where b_k is a subset of S, h_k is a function on b_k to S, and μ_k is a positive constant.

This model of network activity is readily shown to be equivalent to the matrix Q. Observe that each event ξ_k describes a set of transitions having a particular intensity, and thus can be viewed as precisely describing an elementary transition matrix Q_k having a very special form. The set b_k identifies the rows of Q_k which have nonzero entries, the function h_k defines the column $h_k(r)$ in which a single nonzero off-diagonal entry of the rth row ($r \in b_k$) appears, and the constant μ_k defines the value of that element as well as the negative of the diagonal entry of the rth row (row sums must be zero). The matrix Q can be constructed from Ξ by summing

$$Q = \sum_{k \in K_1} Q_k . \tag{5.11}$$

Furthermore, since any matrix Q can be written as a finite sum of such matrices Q_k, one can always construct a (non-unique) set Ξ from any Q. (Of course, the events so constructed will not always have a useful

physical interpretation unless the component Q_k's are chosen well.) This shows the equivalence between Ξ and Q.

The event set Ξ is the intermediate form which is the object of network translation and the subject of numerical analysis. The matrix structure is a computer representation of this set.

6. A NETWORK TECHNOLOGY

At this point the fourth question, "What is the useful information indirectly conveyed by the symbols of the diagram?" will be discussed in general terms. The physical connection between element and network, between the whole and its parts, will be explored. A more detailed model, under specific limitations of application, will be deferred to Section 8.

In the previous section, we have seen that a network can be described by a state set and an event set. Our intuition tells us that individual elements of the network ought also to have things like states and events, or how else could we refer to the "length" of a queue, or the "occurrence of an arrival" in a source element with such disdain for the identification of their neighbors? In this section a generalized interpretation of elements and connections in such intuitive terms will be explored, so that algebraic definitions for element-types and connection-types can bear a realistic interpretation. The algebraic

definitions, which will be developed in Section 8, will be more specific and limited than these interpretations, just as the $\langle I_j, Y_j \rangle$ definition for electric branch types applies to much more restricted electric networks than a discussion of Kirchoff's laws would require.

To be concrete, let us first describe what we consider a "server" to be. It is an object which can be in one of three possible states: "idle," "busy," or "completed." This state can change in three possible ways: (1) If the server is in the "busy" condition, a "service completion" may occur. (2) If the server is in the "idle" condition, an "input" may occur at its input port. (3) If the server is in the "completed" state, an "output" may occur. The first of these is regarded to occur at random times which are determined within the server itself, without regard to occurrences elsewhere in the network containing the server. The second and third occur at times determined by the element or elements connected to the server at the input and output ports, respectively. (These latter elements will be called the <u>associates</u> of the server at the respective ports.) Thus the occurrences which produce changes are more fundamentally of only two types: those which are self-generated (or <u>autogenous</u>) and those which are externally generated (or <u>exogenous</u>).

The change in state resulting from a "service completion" depends upon the condition of its input and output associate. If the input associate "can supply an output," and the output associate "can accept an input," then the service completion will immediately cause both an input to its output associate and an output from its input associate to occur, but will remain in the "busy" state. In other words, it has produced exogenous occurrences for both associates, but its own state hasn't changed. On the other hand, if the output associate could not "accept an input," the server would change to the "completed" state, waiting until the output associate requests an output (i.e., produces an exogenous occurrence at the output port). If the input associate cannot "supply an output" but the output associate can "accept an input," then the server will change to the "idle" state, and cause an input to the output associate to occur.

An "input" occurrence or an "output" occurrence resulting from an external cause (i.e., exogenous occurrence) produces changes in state which are similarly dependent upon properties of the associates. An "input" will produce a change from "idle" state to "busy." An output when the input associate "can supply an output" results in change from "completed" to "busy"; otherwise the change is from "completed" to "idle."

The above description is a slightly abstracted version of our normal concept of the properties of a server. Other interpretations in terms of task flow, or similar notions, will be equivalent and perhaps more easily followed. From this illustration it should be easier to abstract further the principles which will be common to all elements.

First, elements relate to one another only through variables identified with their ports. These variables must have an interpretation which is the same for every element, so that any element may be used as an associate of another without alteration of the definition of either. These variables will be referred to as endoconditions at the respective ports. They represent all of the information about an element which is needed to describe its associate. If the associate is a server, the information needed must tell us, at least, whether or not the element is able to "accept an input" or "supply an output" (depending on whether the port in question is input or output). Depending upon how broad the class of networks to be represented, the endocondition may be interpreted in different ways. If multiple task transfers between elements are possible (as when a bulk server can be represented), the endocondition might describe "the number of tasks which can be accepted (supplied) at an input (output) port." Where networks are restricted by limitations on allowed vocabulary or syntax to one-

task-at-a-time transfers, then a simple two-valued endocondition suffices.

Second, each element must have a set of states which adequately describes its behavior. In the server, this set has three members. In the queue it has as many members as there are allowed lengths of the queue. The state set of the network is contained in the Cartesian product of each element state set. The element state is a projection of the network state. The endocondition is a function of the state of the element, and sometimes must be a function of the endoconditions of the associates as well.

Third, each element has a set (perhaps empty) of objects which will be called autogenous events. These are descriptions of the autogenous occurrences and the state changes that result from them. The state change is generally a function of the state of the element and of the endoconditions of the associates at all the ports. (Although there is only one autogenous event in the server, there are elements which have more than one, hence our reference to a set of them. A queue has no autogenous events.) Typical autogenous events represent service completions in a server, and arrivals in a source element.

Finally, each element will have exogenous events associated with its ports. These describe the exogenous occurrences and the state changes that result from them. As in the case of autogenous events, the state changes are functions (perhaps random) of both the state and the endo-conditions of the associates at the other ports. Typical exogenous events represent the act of inputting a "task" or a "customer" at an input port or of outputting one at an output port. (Notice that logically both of these acts appear to be instances of a single type of object, an exogenous occurrence, and that only the words we use in the interpretation are different.)

In some classes of networks, these two exogenous occurrences are the only ones which can occur. In other classes, such as those including bulk service elements, one can have an input of one task, two tasks, or three tasks, etc. In that case one must identify which of the possible occurrences, determined by the associate, has occurred. To this purpose we must identify another variable, designated an endocontrol, which identifies which of the possible occurrences is occurring at a particular port; for example, "the number of tasks being transferred through the port," would be an interpretation of an endocontrol. The exogenous events of an element at a port will also be functions of this endocontrol of the

associate at that port. When this device is needed, every autogenous and exogenous event must define the endocontrols at the ports at the same time it defines the transitions.

Connections other than "simple" connections serve as a device to alter the identification of the endoconditions of associates to the variables which events depend upon. For example, if a server is connected to two other elements through a "priority branch" connection at its output, then as far as the server is concerned the associates can "accept an input" as long as either one (or both) of them can accept an input, because if the high priority path cannot accept an input, the low priority path is polled. Similarly, the non-"simple" connections alter the identification of the endocontrols. The number of tasks transferred into an input port which is connected to a low priority path of a priority branch connection is the number provided by the supplying associate, less the number transferred into the high priority associate.

The above discussion implies that the states and events of the network can, through the suitable introduction of interface variables (endoconditions and endocontrols), be broken up into pieces, each of which is completely described as a property of an individual element without consideration of that element's context in the full network. Every network event describes the result of a

chain of element events originating from a single autogenous event in one element and propagating to its associates through the effect of exogenous events at the joined ports. The exogenous events in turn involve the associates of the associates the same way, until finally the entire influence of the autogenous event has been traced out.

That states and events of the network can be broken up in this way will, in the last analysis, be demonstrated only by the consistency and interpretability of detailed algebraic systems defined for specific classes of networks using this notion.

7. EQUIVALENCE AND CONSOLIDATION

The interpretation used above represents a viewpoint which is the basis for more precise definitions of the entities described by queueing networks and their components. Much remains to be done in formalizing it, and developing theorems about and classifications of such networks. However, it is not necessary to have a complete, general theory before useful algebraic definitions, especially for specialized subclasses of networks, can be devised and translators developed. Nevertheless, to be useful these specialized definitions must have properties which are consistent with concepts intuitively underlying all network theory.

One of these concepts is the concept of equivalence of subnetworks and networks. Out of this concept will come, almost incidentally, a basic scheme for transforming the network diagram to the matrix structure, and providing an answer for the third of our basic questions. The basic notion of equivalence says that it should be possible to take any repeatable fragment of a network diagram and call it a new type of network element. For example, any of the objects in the boxes in Figure 7.1 should be suitable elements, with the ports shown, if the primitive components are. Of course, the graphics would not necessarily be those shown, and would probably be much more abbreviated.

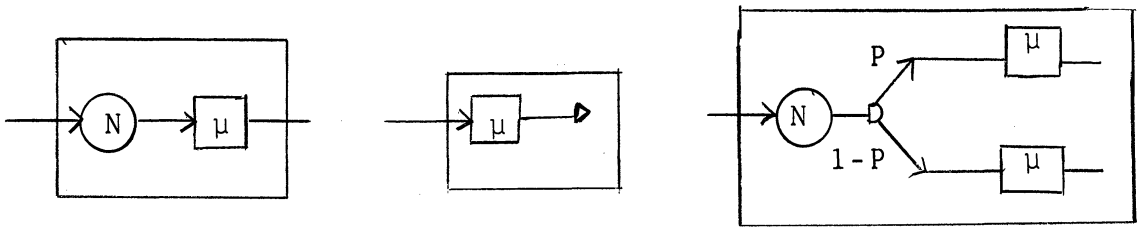

Figure 7.1

Some Nonprimitive Elements

The network resulting from replacing these fragments or "subnetworks" by elements of newly defined element-type should then be equivalent in the sense that the same state set S and event set Ξ should describe it. Thus, Figure 7.2 shows a network which is equivalent to that of Figure 2.1, but uses one of the new elements

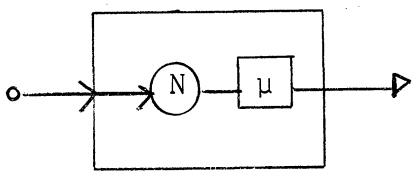

Figure 7.2

An Equivalent Network

of Figure 7.1.

As a consequence of this view, there must be an operator which maps connected sets of elements into "equivalent" elements by "absorbing" the connection which joins them. We have named this the operation of consolidation.

Any definition of element types must be such that this operation can be defined for all syntactically allowed combinations of connections and elements and the result should also be an element. Thus, in an abstract sense, the element types must have such a form that the set of all possible element-types is closed under the operation of consolidation. This is an important property of the algebraic models, and offers a considerable variety of the element types.

A consequence of this property is that the network itself must have an equivalent element: one which contains no ports. Indeed, such an element must constitute a canonical form for a class of equivalent networks derived by various applications of the consolidation operator to the original network. That canonical form may be found by repeated application of the operator until no connections remain.

This procedure describes a potential procedure for the translation of the network diagram to the matrix structure. The consolidation operator is repeatedly applied until only one element remains. This element e* has no ports, hence no exogenous events and endo- or exo-conditions. It will have only a state set S^* and an autogenous event set Ξ^*. With suitable definitions, these two quantities should be readily identifiable with, if not identical to, the state set S and event set Ξ of the network.

8. A MODEL FOR THE SYMBOL SEMANTICS

The network technology discussed above was intended to provide a framework in which to develop models for the element-types and connection-types. These models would represent an answer to the fourth and last question, explicitly describing the information implicitly provided by the user. They would also be of such a form that the consolidation operator was well-defined, or that the translation process could be otherwise described as an algebraic operation upon a known set of operands, analogous to the form $\langle i_j, y_j \rangle$ of branch-types in electric networks.

Only one such model will be described here. This model will be a relatively simple one specifically designed to represent networks containing (at least) queues, simple servers, infinite sources, and exits, with connections including simple, random branch, and priority branches. Of course, in view of the equivalence properties posed in the previous section, many other element-types will also be representable either because they are equivalent to a network fragment containing only the above primitive components, or because of inherent (albeit accidentical) extra generality built into the model. The use of limited models of this type is analogous to the limitations (linear, finite, no-voltage-

source, etc.) which are usually placed on electric network models. The justification is solely one of compromise between simplicity and utility.

An element-type τ_i, $i \epsilon I_1$, has a <u>value</u> which is a triple consisting of a state set S_i, an autogenous event set Ξ_i, and an exogenous event <u>function</u> Z_i

$$\tau_i = \langle S_i, \Xi_i, Z_i \rangle , \quad i \epsilon I_1. \tag{8.1}$$

Each of these will now be defined in greater detail.

The state set S_i of an element e_i is a finite set of nonnegative integer n-tuples, where the dimension of the n-tuples is characteristic of the element-type. For each of the four primitives it is one-dimensional.

By definition, an autogenous event ξ_ℓ of an element e_i having state set S_i is taken to be a triple

$$\xi_\ell = \langle b_\ell, g_\ell, \mu_\ell \rangle \tag{8.2}$$

where

b_ℓ is a subset of S_i

g_ℓ is a constant n-tuple, such that for each $x \epsilon b_\ell$, $x + g_\ell$ is in S_i

μ_ℓ is a positive real number

ℓ is an index in an index set L_i.

Here b_ℓ is called the autocondition set of the event ξ_ℓ, and represents a set of states for which the event is possible; the constant μ_ℓ is called the rate of the event, and represents the probability intensity of its occurrences; the constant n-tuple g_ℓ is called the increment, and represents the change in state which results from the occurrence of the event. (Notice that the change in state has been here specialized to being dependent only on state, not on exoconditions, and being a constant increment.)

To illustrate, consider the "service completion" event ξ_{a_1} for a server having a mean service rate (parameter) γ. The event can occur only in the busy state, state 1, and when in that state it occurs with probability intensity γ. The event causes a change from state 1 to the holding state, state 2, so that the change is unity. Thus,

$$\xi_{a_1} = \langle\{1\}, 1, \gamma\rangle. \tag{8.3}$$

The autogenous event set Ξ_i consists of a set of autogenous events

$$\Xi_i = \{\xi_\ell : \ell \epsilon L_i\}. \tag{8.4}$$

By definition an exogenous event ζ_m at a port p of an element e_i having a state set S_i is taken also to be a triple

$$\zeta_m = \langle b_m, g_m, \pi_m \rangle \qquad (8.5)$$

where

b_m is a subset of S_i

g_m is a constant n-tuple such that for each $x \epsilon b_m$, $x + g_m$ is in S_i

π_m is a probability

m is an index in some index set.

Here b_m is the endocondition set, and represents the set of states for which the endocondition is nonzero (e.g., if the port p is an input port, b is the set of states for which an input of a task can be accepted); the probability π_m is the probability, given that the stimulus (e.g., an input of a task) has occurred and that the element is in a state of b_m, that the change in state produced is equal to g_m, the increment of the event.

An example of an exogenous event is the "input of a task" event (call it ζ_{a_2}) at the input port p of a queue whose maximum length is N. An input can be allowed only when the state is less than N, and it results, with certainty, in an increase of the state by unity. Thus

$$\zeta_{a_2} = \langle\{0,1,\ldots,N-1\},1,1\rangle \qquad (8.6)$$

for $N \geq 1$.

For every port p of an element e_i there is an exogenous event set $Z_i(p)$, representing all the possible responses to the external stimulus at the port p . The function Z_i has been called the exogenous event function, and it is a function on P_i to a set of triples like Eq. 8.5. The set $Z_i(p)$ is a set of exogenous events whose probabilities add up to zero or one for each state of the element. Let

$$Z_i(p) = \{\zeta_m : m \in M_i(p)\}, \quad \text{all } p \in P_i, i \in I_1, \qquad (8.7)$$

where the index sets $M_i(p)$ are disjoint for all p and i . Then

$$\sum_{\substack{m: \\ x \in b_m}} \pi_m = 0 \text{ or } 1 \qquad (8.8)$$

for each $x \in S_i$. Those states for which this sum is zero are states for which the external stimulus cannot occur, such as, for example, the states where an input task cannot be accepted.

Table 8.1 verifies that each of the primitive element types can be represented by this model. A verification that the set of types so-defined is closed under the operation of consolidation is not easily shown,

Table 8.1

Summary of Basic Elements' State Sets and Events

Element Type	Parameter Value	State Set	Autogenous Events	Exogenous Event Function Port	Exogenous Event
Queue	N	{0,1,...,N}	None	p_1	$\begin{cases} \langle\{0,1,\ldots,N-1\}, 1,1\rangle, & \text{for } N\geq 0 \\ \text{None}, & \text{for } N=0 \end{cases}$
				p_2	$\begin{cases} \langle\{1,2,\ldots,N\}, -1,1\rangle, & \text{for } N\geq 0 \\ \text{None}, & \text{for } N=0 \end{cases}$
Server	γ	{0,1,2}	$\langle\{1\},1,\gamma\rangle$	p_1	$\langle\{0\}, 1, 1\rangle$ $\langle\{2\}, -2,1\rangle$
Exit	None	{0}	None	p_1	$\langle\{0\}, 0,1\rangle$
Source	None	{0}	None	p_2	$\langle\{0\}, 0,1\rangle$

and will be accepted here without proof. Moreover, the operation of consolidation will not be defined although it, in fact, constitutes a definition of the connection types. The reader is referred to ref.[3] for a more complete discussion of consolidation. It will merely be noted that, if every connection has been eliminated from the network by the process of consolidation, as proposed in Section 7, the result will be a network consisting of a single element whose port set is empty, hence whose exogenous event function is a function whose domain is empty. Thus the type information merely describes a set of states and a set of autogenous events, the latter being, in fact, the events required in the matrix model of Section 5.

9. CONCLUSIONS

The models required to produce a system for conversational design of a class of stochastic service systems have been described, along with much of the framework of attitudes and viewpoints which are necessary. This represents a start, and was intended to show the nature of the work involved in the creation of such a system. It also represents a good example of the problems likely to be encountered in virtually any network-oriented (i.e., symbolic, graphical) conversational programming system, and the philosophies necessary to their solution.

We have shown how the characteristics of an algebraic network model for Markovian queueing systems are related to the needs of a programming system for conversational design of stochastic service systems. This model represents the beginnings of a mathematical system upon which a useful theory can be built. It also outlines, abstractly, the data structures and procedures of useful programming systems, one example of which is described in [3].

Within this framework, major portions of a specialized model for simple queueing networks have been described (Section 8). This model can be used to describe and translate a wide array of useful queueing networks, and to permit them to be numerically solved. Nevertheless, it represents only

one such model which can be constructed within the framework developed. There are many meaningful elements which were not representable in the simple model, but which can be described in terms of functional relationships among the conditions, controls, and events. An example of an element which cannot be represented in the model of Section 8 is bulk server, which completes tasks in groups of varying size. Models which permit such an element, and others, to be represented must be developed separately as experience with the present model is gained.

REFERENCES

1. Coffman, E. G., Stochastic Models of Multiple and Timeshared Computer Operations, Report No. 66-38, Department of Engineering, U.C.L.A., June 1966.

2. Fife, D. W., "An Optimization Model for Timesharing," Proc. AFIPS Spring Joint Computer Conference, Vol. 28, 1966, pp. 97-104.

3. Irani, K. B., and Wallace, V. L., A System for the Solution of Simple Stochastic Networks, Technical Report 31, Systems Engineering Laboratory, Department of Electrical Engineering, University of Michigan, Ann Arbor, 1968; also Technical Report 14, Concomp Project, Computing Center, University of Michigan, Ann Arbor, 1968.

4. Pinkerton, T. B., Program Behavior and Control in Virtual Storage Computer Systems, Technical Report 4, Concomp Project, Computing Center, University of Michigan April 1968, 160 pp.

5. Scherr, A. L., An Analysis of Timeshared Computer Systems, Project MAC-TR-18, Massachusetts Institute of Technology, Cambridge, June 1965.

6. Smith, J. L., "Multiprogramming Under a Page-on-Demand Strategy," Communications of the ACM, Vol. 10, No. 10, October 1967, pp. 636-646.

7. Sutherland, W. R., The On-line Graphical Specification of Computer Procedures, Ph.D. Dissertation, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, 1966.

8. Wallace, V. L., and Rosenberg, R. S., RQA-1, The Recursive Queue Analyzer, Technical Report 2, Systems Engineering Laboratory, Department of Electrical Engineering, University of Michigan, Ann Arbor, February 1966, 90 pp.

9. Wallace, V. L., and Rosenberg, R. S., "Markovian Models and Numerical Analysis of Computer System Behavior," Proc. AFIPS Spring Joint Computer Conference, Vol. 28, 1966, pp. 141-148.

UNIVERSITY OF MICHIGAN
3 9015 03627 7336