CONSTRUCTION AND SOLUTION OF PERFORMABILITY MODELS BASED ON STOCHASTIC ACTIVITY NETWORKS

by

William Harry Sanders

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1988

Doctoral Committee:

Professor John F. Meyer, Chairperson
Professor Keki B. Irani
Professor Arch W. Naylor
Professor Kang G. Shin
Assistant Professor Mandyam M. Srinivasan

To my parents

ACKNOWLEDGEMENTS

An undertaking such as this can never be accomplished in isolation; it is only through the support and advice of many others, both technical and moral, that it can be carried out. I would first like to acknowledge Professor John F. Meyer, my dissertation advisor. Professor Meyer taught me a way of approaching, analyzing, and solving problems that is invaluable. He inspired me, both in the classroom and through our technical discussions, to do my best at whatever task was at hand. Thanks also go to the remainder of my doctoral committee, Professors Keki B. Irani, Arch W. Naylor, Kang G. Shin, and Mandyam M. Srinivasan. Their collective technical guidance added much to the content of this work.

Many others also contributed in various ways to the dissertation. In particular, I'd like to thank the past and present members of the Communications and Distributed Systems Laboratory at the Industrial Technology Institute, for both technical and financial support during the last four and a half years. It was here that many of the ideas of this dissertation were formulated and tested. Special thanks go to those who helped develop METASAN™, a performability evaluation tool based on many of the early ideas of this dissertation: Paul Daugherty, Jorge Goić, Steve Sparks, and Greg Ullmann. Their hard work was essential in making the tool a reality. Particular thanks go to Jorge, whose timely implementation of some of the reduced base model construction techniques developed herein permitted them to be tested on realistic systems. I would also like to thank Ali Movaghar, for the development of some of the basic ideas regarding stochastic activity networks, and K. H. Muralidhar, for his sound technical advice and collaboration on the token bus network evaluation study presented in the applications chapter.

Heartfelt thanks also go to those who read early versions of this manuscript and made helpful comments, both technical and editorial, which improved its contents and presen-

tation style. In addition to those mentioned earlier, I would like to thank Linda Atwood, Emily Churchill, and Lola McGuffin for their help in this capacity. I would also like to thank Emily Churchill and Michelle Schiller for their help in producing some of the illustrations in this document.

Lastly, but by no means lesser in importance, I would like to thank my family and friends for their moral support during the time of this effort. While technical advice is necessary, it is by no means sufficient. Without their encouragement, none of this work would have been possible.

TABLE OF CONTENTS

DICTIONARY OF PRINCIPAL SYMBOLS

Symbol		Page
μ	a marking	13
$I\!N$	the natural numbers	13
M_S	possible markings of a set of places S	13
e	an enabling predicate	13
f	an input function or output function	13
AN	an activity network	13
P	a finite set of places	13
A	a finite set of activities	13
I	a finite set of input gates	13
O	a finite set of output gates	13
γ	the case function of an AN	13
τ	the type function of an AN	13
ι	the input structure function of an AN	13
o	the output structure function of an AN	13
μ_S	the restriction of a marking to a set of places S	14
$IP(a)$	input places for an activity a	14

LIST OF DEFINITIONS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

INTRODUCTION

Background

System evaluation can be done during specification, design, implementation, and modification of a system, and by different methods. Each of these phases presents specific challenges and, accordingly, several different approaches to evaluation have evolved. Measurement can be an effective tool when the system to be studied has been implemented. This approach typically entails using hardware and/or software monitors to collect information regarding the operation of the system and the use of statistical methods to estimate performance indices. Several disadvantages are associated with exclusive use of measurement. First, the system to be evaluated must be implemented. This precludes evaluation during the specification and design phases of a project. Second, this type of evaluation is typically costly, since it requires the development of special data collection hardware and/or software and use of the implementation to collect the data. In addition, to evaluate a proposed change to a system, it must be implemented before it can be evaluated, typically resulting in much down-time and cost.

In order to avoid the drawbacks inherent with measurement, system modeling has been used extensively. Instead of collecting data from the actual system, a model is constructed and performance and/or dependability characteristics are estimated from the model. This type of evaluation has several advantages over measurement. First, a model can be constructed during any phase of the life-cycle of a system. Second, evaluation via

modeling is typically much less costly, both to develop and to use. In order to study the effect of a change in design, one need only change the model.

Traditionally, model-based evaluations have been accomplished by treating *performance* [23,36,49] and *dependability* [2,48] as separate issues, under the assumption that a system either performs as expected (delivers its expected service) or fails. In such cases, performance means "failure-free performance" and dependability means "the ability to perform successfully". While these separate approaches were sufficient for systems in the past, they do not suffice for distributed systems, which pose challenging evaluation problems, due to both their complexity and the application environments in which they operate. Properties that contribute to system complexity include concurrency, fault tolerance, and degradable performance. In the case of real-time environments, timeliness is an additional consideration. Accordingly, as first noted in [65], all of these properties should be accounted for in the evaluation process. In particular, degradable performance calls for unified performance-dependability measures, which, in our terminology, quantify a system's *performability* [63,64].

Performability

Performability and its associated concepts are defined as follows. Let S denote the system in question where S is generally interpreted as including not only a system, *per se*, but also relevant aspects of its environment (e.g., workload, external sources of faults, etc.). Then the *performance* of S over a specified *utilization period* T is a random variable Y taking values in a set A. Elements of A are the *accomplishment levels* (performance outcomes) to be distinguished in the evaluation process. The *performability* of S is the probability measure $Perf$ (denoted p_S in [63,64]) induced by Y where, for any measurable set B of accomplishment levels ($B \subseteq A$),

$$Perf(B) = \text{ the probability that } S \text{ performs at a level in } B.$$

Solution of performability is based on an underlying stochastic process X, called a *base model* of S, which represents the dynamics of the system's structure, internal state, and environment during utilization. By its definition, the base model must also "support" the

solution of $Perf$ in the sense that, for any accomplishment set B of interest, $Perf(B)$ can be formulated in terms of X. A base model X together with a performance variable Y is a *performability model* of S. Performability *model construction* is the process of identifying a performance variable Y and determining a base model stochastic process X that permits the solution of performability. Performability *model solution* is the process of obtaining performability values $Perf(B)$ for accomplishment sets B that are of interest to the user.

Stochastic Extensions to Petri Nets

While the concept of performability provides a framework to delineate the scope of the evaluation activity, it in itself does not suggest methods for model construction and solution. Queueing networks have been used extensively for the evaluation of computing systems [36,45], but their application to systems that exhibit complex concurrency, fault tolerance, and degradable performance is not straightforward. Extensions to Petri nets [76], on the other hand, have proved to be valuable tools for evaluation of systems that exhibit these properties. Many of these extensions have included the addition of an explicit representation of time. In particular, one approach has been to add timing of a probabilistic nature to transitions in the net. This permits the representation of both performance and dependability related characteristics, depending on the interpretation given to the tokens in the model.

Early work whose aim was to add probabilistic timing to the transitions of the net can be traced back to three independent efforts each begun in the late 1970's [69,73,82]. All of these efforts shared the common idea of associating time with the transitions of classical Petri nets, but differed in details. The most well known is the work of Molloy [69] and of Natkin and Florin [73]. In each of these efforts, the authors augmented standard Petri nets by associating an exponentially distributed delay with each transition in the standard Petri net (Molloy also considered geometrically distributed time). Then, by assigning an appropriate interpretation to the execution of the net, they noted that the sequence of markings that arises during an execution of the net can be represented as a continuous-time discrete-state Markov process (discrete-time discrete-state in the case of geometrically

distributed delays). This observation permitted the authors to solve for a variety of performance variables in terms of the solution of the associated Markov process. Standard Markov analysis techniques were used. Examples include determining the throughput of an alternating bit protocol via a steady-state state occupancy probability solution [69] and determining the dependability of a subway system by determining the mean time to absorption of an associated Markov process [8]. Shapiro [82] added time to the transitions of the Petri net in a similar manner, but required all places in the net to be 1-safe (have a marking of 0 or 1). This led to a process where the marking of each place could be modeled by an indicator random variable.

These early efforts were quite successful in both representing and solving for the behavior (both dependability and performance related) of complex concurrent systems, but it was soon realized that it was possible to extend them in order to enhance the representation power and modeling convenience of the networks. There were three basic driving forces in this regard. First, it was realized that the models should be extended, at least conceptually, to include cases where the delays associated with the transitions in the net were not exponential. Subtleties masked by the memoryless property of the exponential distribution then became apparent. This was handled in different ways by different researchers. Second, it was noted that the difference in magnitude of the various rates associated with the transitions in the net was typically large. Assuming that the time spent in markings in which very fast transitions are enabled is negligible (in terms of the desired performance variables), they could be replaced by transitions that had a zero delay time, with an interpretation that reduced the state space size of the associated Markov process. Third, it was noted that additional constructs could be added to the net structure to facilitate representation of systems in a compact manner, while not changing the probabilistic behavior of the net.

As was the case with the initial attempts to add time of a stochastic nature to Petri net transitions, several groups of researchers produced somewhat similar independent extensions at approximately the same time. The first generalization to appear was that of Marsan, Balbo, and Conte [56,57]. Marsan *et al.* introduced "Generalized Stochastic Petri Nets" (GSPNs) and proposed their use as stochastic models for the performance

evaluation of computer systems which exhibit synchronization, passive resources, and/or blocking. A brief review of their generalizations follows. First, *immediate* transitions were introduced to model events in a system that take a negligible amount of time. In the case of multiply enabled immediate transitions, a *random switch* is used to determine the probability that each enabled transition fires in a given marking. An execution of a GSPN consisted of a sequence of transitions between a set of *tangible* and *vanishing* states. Tangible states represent markings in the GSPN that are occupied for a non-zero amount of time before reaching a next state, while vanishing states represent markings in which no time is spent before the next transition. A model construction method was proposed that eliminated vanishing states, thus reducing the size of the state space required (compared to initial SPN formulations) for a given problem. Two software tools to aid in model construction and solution using GSPNs have been presented: the first by Marsan, Balbo, Giardo, and Conte [55] and the second by Chiola [12]. The general approach taken by these packages is to construct a stochastic process representation of the execution of the GSPN and then solve this representation for desired system characteristics using known methods. The second package [12] also had a provision for simulating the behavior of the net when it was not analytically tractable. GSPNs have been used as a modeling framework in many modeling studies. Examples include studying the performance/reliability of degradable multi-processor systems [58], the effect of software blocking [3,4], the effect of different queueing disciplines [5], and the performance of a CSMA/CD protocol [60]. The last study made use of a solution technique [59] that allowed for the solution of a limited class of GSPNs that have deterministic or exponentially distributed delays associated with transitions.

A second attempt to generalize stochastic Petri nets was made by Dugan, Trivedi, Geist, and Nicola [22]. They introduced the notion of an "Extended Stochastic Petri Net" (ESPN) to model systems similar to those modeled by GSPNs. ESPNs differ from GSPNs in several respects. First, the ESPN formalism allows the delays associated with transitions to be represented by arbitrary distribution functions. In addition, a *probabilistic arc* was introduced. Its interpretation is as follows. A probabilistic arc from a transition to a set of output places deposits a token in one of the places in the set according to some

fixed probability distribution. *Counter arcs* and *counter-alternate arcs* were also defined. Although these constructs added no modeling power to the nets, they added convenience. The remainder of Dugan's work [21] consisted of defining conditions when a given ESPN represented a Markov or semi-Markov process, describing a software package based on ESPNs (DEEP), and applying ESPNs to several modeling problems.

A third independent extension was introduced by Movaghar and Meyer [71] under the name "stochastic activity networks" (SANs). *Stochastic activity networks* consist of activities, places, input gates, and output gates. Activities ("transitions" in Petri net terminology) are of two types, timed and instantaneous. Elongated ovals represent *timed activities* and solid bars represent *instantaneous activities.* Timed activities are used to represent activities of the modeled system whose durations impact the system's ability to perform. Instantaneous activities (immediate transitions in GSPN terminology) represent system activities which, relative to the performance variable in question, complete in a negligible amount of time. Places are depicted as circles and, as with Petri nets, each place can hold a nonnegative number of *tokens.* The distribution of tokens in the places of the network at a given time constitutes the *marking* of the network at that time.

Cases (a generalization of probabilistic arcs) can be associated with both timed and instantaneous activities and are represented by small circles. Cases permit the realization of two types of spatial uncertainty. Uncertainty about which activities are enabled in a given marking is realized by cases associated with intervening instantaneous activities. Uncertainty about the next marking assumed upon completion of a timed activity is realized by cases associated with that activity. Input gates contain both an enabling predicate and input function (on the marking of the places). The enabling predicate must be true for the activity associated with that gate to be enabled. Upon completion of the associated activity, the input function is executed, possibly changing the marking of the net. Output gates have a single output function (on the marking of the places) associated with them, which is executed upon completion of the associated activity.

The stochastic nature of the nets is realized by associating an activity time distribution function with each of the timed activities and a probability distribution with each set of cases. A *reactivation function* [66] is also associated with each timed activity. This

function specifies, for each marking, a set of *reactivation markings.* Informally, given that an activity is activated in a specific marking, the activity is *reactivated* whenever any marking in the set of reactivation markings for the specified marking is reached. This provides a mechanism for restarting activities that have been activated, either with the same or a different distribution. Note that this decision is made on a per activity basis (based on the reactivation function), and is not a net-wide execution policy. Meyer, Movaghar, and Sanders [66] presented conditions under which the behavior of a SAN is a Markov or semi-Markov process as well as a SAN-based performability evaluation procedure.

Other generalization attempts have also been made. In particular, SPNs with phase-type [74] delays associated with transitions were considered by Bobbio and Cumani [9], and implemented in the software tool ESP [17]. SPNs with general delay time distributions were studied by Marsan, Balbo, Bobbio, Chiola, Conte, and Cumani [53]. They defined formally the notion of an "execution sequence", and discussed different "execution policies". In a broad sense, certain execution policies can be thought of as specific assignments of reactivation functions to activities (in the notation of Meyer, Movaghar, and Sanders). Properties of colored SPNs were investigated by Zenie [91]. Finally, Zuberek studied two types of stochastic Petri nets: D-Nets and M-Nets, which have deterministic and exponentially distributed transitions, respectively [92].

Several types of stochastic Petri nets have been considered with the express intention of solution by simulation. The most rigorous investigation has been done by Haas and Shedler [34] who derive conditions which ensure that the stochastic process associated with the SPN is a regenerative process in continuous time with a finite expected time between regeneration points. This work was later utilized by Prisgrove and Shedler [77], who investigated discrete-event simulation of symmetric SPNs. "Evaluation nets" are considered by Behr *et al.* in [7] where they describe the simulation package FORCASD. Godbersen and Meyer [27] have introduced "function nets" which allow for performance evaluation via simulation and formal (non-time related) analysis via classical Petri net techniques. Finally, simulation of SPNs in Simula has been done by Törn [86].

Research Objectives

While stochastic extensions to Petri nets provide a natural representation for systems we are interested in, model construction and solution techniques for them are not as well developed as they are for other model types (e.g., queueing networks). In particular, when the intended solution method is analytic, model construction has consisted of taking the possible reachable stable (or tangible) markings of the network to be states of the process. Performance variables are then written at the state level in terms of this process, and do not refer explicitly to behaviors at the network level.

The use of stochastic Petri nets has thus been limited to providing a convenient representation of either a Markov process or simulation model, and the structure of the net and performance variable has not been used to aid in model construction or solution. Two general problems arise from this limited approach: 1) performance variables are limited to those that can be written in terms of the "marking space" of the stochastic Petri net, and 2) construction techniques result in a large state-space representation (regardless of the minimal base model that is required to support the chosen performance variables) which grows rapidly with the size of the net.

The goal of this investigation is, then, to develop a coherent set of construction and solution methods that exploit, to as great an extent as possible, the structure of the net in the modeling process. Stochastic activity networks will be used as the extended Petri net representation. This approach will allow performance variables to be specified at the network level, and will allow the nature of these performance variables to be used in the construction procedure. More specifically, we will:

1. Provide formal definitions of stochastic activity networks and related concepts and specify when SANs can be used for performability evaluation.

2. Define a general class of performance variables at the SAN level.

3. Develop model construction techniques that make use of these variables and network characteristics to reduce the size of the base model representation.

4. Investigate the use of traditional stochastic process solution techniques to solve for

the defined variables, develop a solution method for expected reward that is applicable to one "type" of the defined variables, and develop a simulation procedure to solve for many variables.

5. Illustrate the use of these methods by evaluating the performability of two distributed systems.

Chapter 2 provides the basic definition of stochastic activity networks, and specifies when they can be used for performability evaluation. In particular, stochastic activity networks are defined in terms of a simpler non-probabilistic model class known as *activity networks*, similar to the manner in which stochastic Petri nets are defined in terms of Petri nets. This definition allows the investigation of conditions under which a SAN is *well specified* or, in other words, the stochastic extension made to the activity network completely specifies the network's probabilistic behavior. This is a necessary condition for the SAN to be used for evaluation.

Chapter 3 defines performance variables that can be written directly at the SAN level. Variables that depend on both activity completions and markings of places are permitted. Expressing variables at the SAN level has two distinct advantages over expressing them at the process level, as has been done previously. First, it allows the analyst to think about the variables at the same level as the structure of the system being modeled, eliminating the need to think in terms of the possibly very large process representation. Second, it permits the variables to be used in the construction procedure, both ensuring that the base model representation is detailed enough to "support" the variable and is no larger than is necessary to be "solvable".

Chapter 4 develops model construction techniques for stochastic activity networks and the variables that are defined in Chapter 3. More specifically, two classes of base models are considered. In particular, a base model which supports a large class of variables is referred to as a *detailed base model* while a base model constructed to support a designated performance variable is a *reduced base model.* Reduced base models, while limited to a particular class of systems, typically have fewer states and, hence, are easier to solve. Conditions defining when reduced base models may be obtained are given, along with a proof that, when these conditions are met, the resulting stochastic process is indeed a base

model (i.e., it supports the variable of interest) and, moreover, permits the variable to be solved for using known techniques. Construction procedures are also developed for both classes of base models.

Chapter 5 investigates the use of traditional stochastic process solution techniques for use in solution of the defined variables and develops a solution method for one "type" of variable defined in Chapter 3. The solution method is applicable to systems that have a finite "lifetime", i.e., there is a time after which no further changes in the variable occur (with probability one). In addition, a procedure is developed to simulate a stochastic activity network at the network level. This permits estimation of performance variables when the nature of the SAN does not allow solution by analytic methods.

Chapter 6 presents the results of the application of the construction and solution techniques developed in this dissertation to the evaluation of two distributed systems. The first is an evaluation of an industrial network employing the IEEE 802.4 token bus protocol. The performance of the network is evaluated in the presence of faults caused by noise bursts and token losses. The second is an evaluation of a CSMA/CD local area network. It both illustrates the state-space savings achieved through the use of reduced base model construction methods and investigates the effect that a particular priority scheme has on the delay that messages experience when trying to access the channel. These two applications illustrate that stochastic activity networks and the construction and solution techniques developed herein are indeed applicable to complicated distributed systems.

Chapter 7 summarizes the results of this dissertation and suggests topics for further research.

An appendix, which describes a software tool for the evaluation of systems using stochastic activity networks, is also included. This tool, METASAN[1], facilitates both construction and solution of systems modeled as stochastic activity networks. METASAN provided an environment in which to test the usefulness of the techniques developed in this dissertation and a tool to evaluate the performability of many systems.

---

[1] METASAN is a Trademark of the Industrial Technology Institute.

CHAPTER II

MODELING FRAMEWORK

Introduction

In order to be effective and generally applicable, model construction and solution techniques require a formal representation scheme. The scheme must be general enough to allow for easy representation of realistic systems, and formal enough to permit derivation of useful results. The modeling framework discussed in this chapter serves this purpose. To account for degradable performance, the concept of performability is employed. To permit the representation of parallelism, timeliness and fault tolerance which may be present in distributed systems, stochastic activity networks are used. These models allow for compact representation of large systems and, due to the chosen definition of execution, exhibit behaviors which may be described in terms of one of several stochastic processes, depending on the structure of the network-level representation and the designated performance variables.

Activity Networks

The desire to represent system characteristics of parallelism and timeliness, as well as fault tolerance and degradable performance, precipitated the development of general network-level performability models known as stochastic activity networks [66,71]. Stochastic activity networks are probabilistic extensions of "activity networks", where the nature of the

extension is similar to the definition of stochastic Petri nets in terms of (classical) Petri nets.

Definitions

Informally (as in [71]), activity networks are generalized Petri nets with the following primitives:

- *activities*, which are of two kinds: *timed* activities and *instantaneous* activities. Each activity has a non-zero integral number of *cases*[1].
- *places*, as in Petri nets.
- *input gates*, which have a finite set of inputs and one output. Associated with each such input gate is an n-ary computable predicate and an n-ary computable partial function over the set of natural numbers called the *enabling predicate* and the *input function*, respectively. The input function is defined for all values for which the enabling predicate is true.
- *output gates*, which have a finite set of outputs and one input. Associated with each such input gate is an n-ary computable function on the set of natural numbers called the *output function*.

Timed activities represent activities of the modeled system whose durations impact the system's ability to perform. Instantaneous activities, on the other hand, represent system activities which, relative to the performance variable in question, are completed in a negligible amount of time. Cases associated with activities permit the realization of two types of spatial uncertainty. Uncertainty about which activities are enabled in a certain state is realized by cases associated with intervening instantaneous activities. Uncertainty about the next state assumed upon completion of a timed activity is realized by cases

[1] The term *case*, as used here, should not be confused with the notion of *cases* of elementary net systems [85]. Here the term case is used to denote a possible action which may be taken upon the completion of an event.

associated with that activity. Gates are introduced to permit greater flexibility in defining enabling and completion rules.

Before formally defining an activity network, it helps to define several related concepts in a more precise manner. Let P denote the set of all places of the network. If S is a set of places ($S \subseteq P$), a *marking of* S is a mapping $\mu : S \rightarrow I\!N$. Similarly, the set of *possible markings of* S is the set of functions $M_S = \{\mu \mid \mu : S \rightarrow I\!N\}$. With these definitions in mind, an *input gate* is defined to be a triple, (G, e, f), where $G \subseteq P$ is the set of *input places* associated with the gate, $e : M_G \rightarrow \{0, 1\}$ is the *enabling predicate* of the gate and $f : M_G \rightarrow M_G$ is the *input function* of the gate. Similarly, an *output gate* is a pair, (G, f), where $G \subseteq P$ is the set of *output places* associated with the gate and $f : M_G \rightarrow M_G$ is an *output function* of the gate. One can then formally define an activity network in terms of allowable interconnections between these model primitives.

Definition II.1 An *activity network* (AN) is an eight-tuple

$$AN = (P, A, I, O, \gamma, \tau, \iota, o)$$

where P is some finite set of places, A is a finite set of activities, I is a finite set of input gates, and O is a finite set of output gates. Furthermore, $\gamma : A \rightarrow I\!N^+$ specifies the number of cases for each activity, and $\tau : A \rightarrow \{Timed, Instantaneous\}$ specifies the type of each activity. The net structure is specified via the functions ι and o. $\iota : I \rightarrow A$ maps input gates to activities, while $o : O \rightarrow \{(a, c) | a \in A \text{ and } c \in \{1, 2, \ldots, \gamma(a)\}\}$ maps output gates to cases of activities.

Several implications of this definition are immediately apparent. First, each input and output gate is connected to a single activity. In addition, each input of an input gate and output of an output gate is connected to a unique place. In contrast to the definition in [71], different output gates and input gates of an activity may be connected to identical places, as has been done in practice. Ambiguity in the execution of the net is avoided by requiring that the marking obtained upon completion of each activity not depend on 1) the order of application of the input gate functions, or 2), the order of application of the output gate functions.

The following definitions aid in the discussion that follows.

Definition II.2 If $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ is an activity network and $S, G \subseteq P$ then

1. a mapping $\mu : P \rightarrow I\!N$ is a *marking of the network*,
2. for $S \subseteq P$, $\mu_S : S \rightarrow I\!N$ is the restriction of μ to places of S (i.e. $\mu_S(p) = \mu(p), \forall p \in S$),
3. an input gate $g = (G, e, f)$ *holds* in a marking μ if $e(\mu_G) = 1$,
4. an activity a is *enabled* in a marking μ if g holds for all $g \in \iota^{-1}(a)$,
5. a marking μ is *stable* if no instantaneous activities are enabled in μ,
6. the *input places of an activity* a is the set $IP(a) = \{p \mid \exists\ (G, e, f) \in \iota^{-1}(a)$ such that $p \in G\}$, and
7. the *output places of an activity* a is the set $OP(a) = \{p \mid$ for some $c = 1, 2, \ldots, \gamma(a)$, $\exists\ (G, f) \in o^{-1}(a, c)$ such that $p \in G\}$.

The marking of a network can alternatively be represented as a vector, where each component of the vector is the number of tokens in a particular place. The correspondence of components of the vector with markings of places is done via some designated total ordering of P. For example, for a set of places $\{p_1, p_2, \ldots, p_n\} \subseteq P$ and marking vector $(n_1, n_2, \ldots, n_n)$, $\mu(p_1) = n_1, \mu(p_2) = n_2, \ldots, \mu(p_n) = n_n$, if $p_1 < p_2 < \ldots < p_n$. The functional notation for markings is more convenient for the development of theory, while the vector notation is useful for examples.

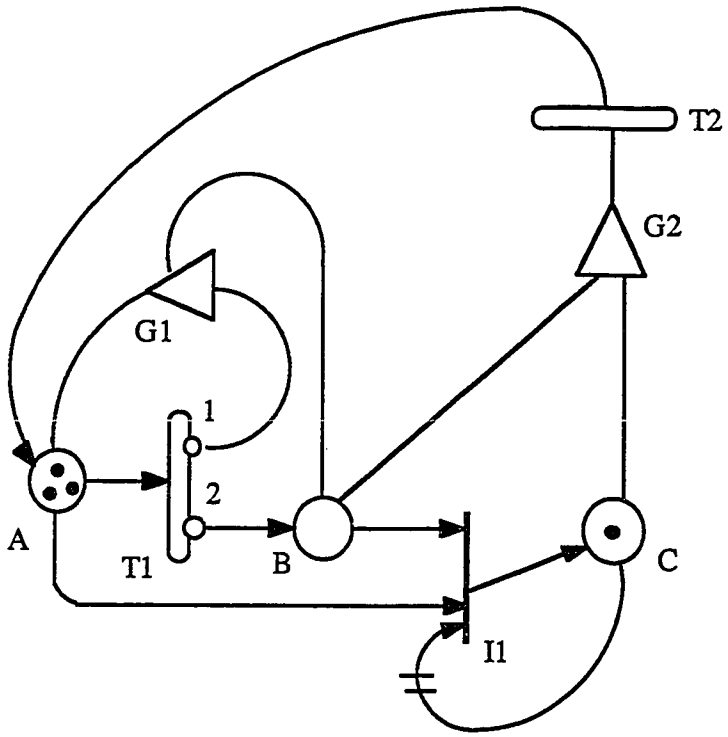
Graphical Representation

To aid in the modeling process, a graphical representation for activity networks is typically employed. In fact, for all but the smallest networks, specification via the tuple formulation presented in the definition is extremely cumbersome. Not only is the graphical representation more compact, but it also provides greater insight into the behavior of the network. For example, let i and j be natural numbers and consider the following activity network

$$P = \{A, B, C\}, \text{ where } A < B < C$$
$$A = \{T1, T2, I1\}$$

$$
\begin{aligned}
I &= \{GA1, GA2, GB, GC, G2\} \\
O &= \{AG, BG, CG, G1\} \\
\gamma &= \{(T1,2),\ (T2,1),\ (I1,1)\} \\
\tau &= \{(T1, Timed),\ (T2, Timed),\ (I1, Instantaneous)\} \\
GA1 &= (\{A\},\ \{(i,1) \mid i>0\} \cup \{(0,0)\},\ \{(i,i-1)\} \mid i>0\} \cup \{(0,0)\}) \\
GA2 &= (\{A\},\ \{(i,1) \mid i>0\} \cup \{(0,0)\},\ \{(i,i-1)\} \mid i>0\} \cup \{(0,0)\}) \\
GB &= (\{B\},\ \{(i,1) \mid i>0\} \cup \{(0,0)\},\ \{(i,i-1)\} \mid i>0\} \cup \{(0,0)\}) \\
GC &= (\{C\},\ \{(i,0) \mid i>0\} \cup \{(0,1)\},\ \{(i,i) \mid \forall i\}) \\
G2 &= (\{B,C\},\ \{((i,j),1) \mid i>0 \text{ or } j>0\} \cup \\
&\quad \{((0,0),0)\},\ \{((i,j),(0,0) \mid \forall i,j\}) \\
AG &= (\{A\},\ \{(i,i+1) \mid \forall i\}) \\
BG &= (\{B\},\ \{(i,i+1) \mid \forall i\}) \\
CG &= (\{C\},\ \{(i,i+1) \mid \forall i\}) \\
G1 &= (\{A,B\},\ \{((i,j),(i+2,j)) \mid i<5 \text{ and } j<5\} \cup \\
&\quad \{((i,j),(i-1,j)) \mid j>5 \text{ and } i=0\} \cup \\
&\quad \{((i,j),(i,j)) \mid j>5 \text{ and } i \neq 0\}) \\
\iota &= \{(GA1,T1),\ (GA2,I1),\ (GB,I1),\ (GC,I1),\ (G2,T2)\} \\
o &= \{(AG,(T2,1)),\ (BG,(T1,2)),\ (CG,(I1,1)),\ (G1,(T1,1))\}
\end{aligned}
$$

Figure 2.1 depicts the graphical representation of this network. One can immediately see the utility of a graphical representation. Here places are represented by circles (e.g., A, B, C), as in Petri nets. Timed activities (e.g., $T1$, $T2$) are represented as hollow ovals. Instantaneous activities (e.g., $I1$) are represented by solid bars. Cases associated with an activity are represented by small circles on one side of the activity (e.g., $T1$). Activities with one case are represented with no circles on the output side (e.g., $T2$).



Gate	Predicate	Function
G1	-	if (MARK(A) < 5 and MARK(B) < 5 then MARK(A) = MARK(A) + 2; else if (MARK(A) > 0) then MARK(A) = MARK(A) - 1;
G2	MARK(B)>0 or MARK(C)>0	MARK(B) = 0; MARK(C) = 0;

Figure 2.1: Graphical Activity Network Representation

Gates are represented by triangles. $G2$ is an example of an input gate with 2 inputs. $G1$ is an example of an output gate with 2 outputs. Enabling predicates and functions for gates are typically given in tabular form. Three types of commonly used gates are given default (non-triangle) representations for ease of interpretation and to illustrate their similarity to classical Petri net primitives. In particular, an input gate with one input, enabling predicate $\{(i,1) \mid i > 0\} \cup \{(0,0)\}$ and function $\{(i,i-1)\} \mid i > 0\} \cup \{(0,0)\}$ (e.g., $GA1$) is represented as a directed line from its input to its output. Similarly, an output gate with one output and output function $\{(i,i+1) \mid \forall i\}$ (e.g., AG) is shown as a directed line from its input to its output. Finally, an input gate with one input, enabling predicate $\{(i,0) \mid i > 0\} \cup \{(0,1)\}$, and function $\{(i,i) \mid \forall i\}$ (e.g., GC) is shown as a directed line from its input to its output crossed by two short parallel lines. This type of input gate corresponds to an inhibitor arc in extended Petri nets. These shorthand notations for gates aid in understanding the behavior of a network from its graphical representation.

Activity Network Behavior

The behavior of an activity network is a characterization of the possible completions of activities, selection of cases, and changes in markings. Specifically,

Definition II.3 An activity a *may complete* in a marking μ if

1. a is enabled in μ, and
2. if a is timed, no instantaneous activities are enabled in μ.

This imposes two explicit priority classes on activities. We can now define the result of the completion of an activity and selection of a possible case. This is made easier by expanding the domain and range of the gate functions to the complete network marking. Specifically, for an activity network with places P, gate (of the network) with set of places G and function f, define the function $\tilde{f} : M_P \rightarrow M_P$ where if $\tilde{f}(\mu) = \mu'$ then

$$\mu'(p) = \begin{cases} f(\mu_G)(p) & \text{if } p \in G \\ \mu(p) & \text{otherwise.} \end{cases}$$

Using this notion, we can define an activity completion and case selection.

Definition II.4 Given an activity network $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ with activity a which may complete in μ, the *completion of activity a and choice of case c in μ yields*

$$\mu' = \tilde{f}_{O_m}(\cdots \tilde{f}_{O_1}(\tilde{f}_{I_n}(\cdots \tilde{f}_{I_1}(\mu)\cdots))\cdots)$$

where $\iota^{-1}(a) = \{I_1, \ldots, I_n\}$ and $o^{-1}(a, c) = \{O_1, \ldots, O_m\}$.

While the gates in the two sets are numbered, there is no implied ordering on their application within a set, since the SAN definition does not specify an ordering among input gates or output gates. Output gate functions are applied after input gate functions, however. The notation $\mu \xrightarrow{a,c} \mu'$ is used to indicate that the completion of a and choice of c *yields* μ'. Furthermore, we say that marking μ' is *immediately reachable* from a marking μ if $\mu \xrightarrow{a,c} \mu'$ for some activity a and case c.

The set of reachable markings from a given marking can be defined directly in terms of the reflexive, transitive closure of the yields relation, which we denote as $\xrightarrow{*}$. Using this notation, the set of reachable markings from some marking can be defined as follows.

Definition II.5 The set of *reachable markings* of an activity network AN in a marking μ_0 is the set of markings $R(AN, \mu_0)$ where

$$R(AN, \mu_0) = \{\mu \mid \mu_0 \xrightarrow{*} \mu\}.$$

Sets of "stable reachable markings" and "unstable reachable markings" of an activity network can then be defined in terms of its reachable markings. Specifically, the set of *stable reachable markings* of an activity network AN in an initial marking μ_0 is the set $SR(AN, \mu_0) \subseteq R(AN, \mu_0)$ of reachable markings of AN from μ_0 that are stable. Similarly, the set of *unstable reachable markings*, denoted $UR(AN, \mu_0)$, is the set of markings reachable from μ_0 that are not stable.

The behavior of an activity network can be described in terms of successive applications of the yields relation. Each application of the yields relation represents the completion of one activity of the one or more activities that may complete in the marking. Note that, unlike elementary net systems [85], the yields relation is defined only for single

activities and that the concurrent completion of more than one activity is not considered. Each step in the evolution of the network is called a *configuration* which, formally, is a marking-activity-case triple $< \mu, a, c >$ where a is some activity with case c which may complete in μ. A *completion of a configuration* occurs when the activity associated with the configuration completes. The behavior of a network can thus be described in terms of possible sequences of configurations, more formally called "paths".

Definition II.6 A *path* of an activity network, AN, with marking μ_0 is a sequence of configurations $< \mu_1, a_1, c_1 >, < \mu_2, a_2, c_2 >, \ldots, < \mu_n, a_n, c_n >$ such that,

1. $\mu_1 \in R(AN, \mu_0)$,
2. for each pair of configurations $< \mu_i, a_i, c_i >$, $< \mu_{i+1}, a_{i+1}, c_{i+1} >$ $(1 \leq i < n)$, $\mu_i \stackrel{a_i, c_i}{\rightarrow} \mu_{i+1}$, and
3. $\mu_n \stackrel{a_n, c_n}{\rightarrow} \mu'$ for some marking μ'.

Definition of several additional terms aid in the discussion that follows. In particular, the *initial marking of a path* is the marking of the first configuration in the path. The *resulting marking of a path* is the marking that is reached upon completion of the last configuration in the path. A path is said to be *from μ to μ'* if μ is the initial marking of the path and μ' is the resulting marking of the path.

Stochastic Activity Networks

Activity networks are interesting in their own right and several properties of them have been studied [70]. However, for the purpose of this dissertation, they serve as a non-probabilistic base for a stochastic extension, called stochastic activity networks, which is used for performability evaluation. When used in this manner, care must be taken to insure that the probabilistic behavior of the stochastic extension is completely specified. Specifically, since we want to be able to ask questions regarding possible sequences of timed activity completions and intervening stable markings, we require that a stable marking eventually be reached after any sequence of consecutive instantaneous activity completions. Identification of situations where this may occur is aided by the introduction of the notion of a "step".

Definition II.7 Let AN be an activity network and s be a path of AN with initial marking μ_0. Then s is a step if:

1. the initial marking of s is stable, and
2. the markings of all other configurations of s are unstable.

Note that it is not required that the resulting marking of the step be stable. The set of markings that can be reached by completion of different steps with a single initial marking provides insight into the behavior of an activity network. To see this, let

$$S(\mu) = \{s \mid s \text{ is a step with initial marking } \mu\}$$

where μ is a stable reachable marking of the AN in question. Now, since there are only a finite number of steps of a given length from any marking μ, the cardinality of $S(\mu)$ is $\aleph_0$ if and only if the length of steps in $S(\mu)$ increases without bound. Or equivalently, since all of the activities except the first in a step are instantaneous, $|S(\mu)| = \aleph_0$ if and only if an unbounded number of instantaneous activities can complete without resulting in a stable marking. This leads us to the following definition of a "stabilizing" activity network. Formally,

Definition II.8 An activity network AN in a marking μ_0 is *stabilizing* if, for every $\mu \in SR(AN, \mu_0)$, the set $S(\mu)$ is finite.

The following example illustrates the concept of stabilizing and non-stabilizing activity networks in a marking. Consider the activity network of Figure 2.2. If we denote its marking as a vector using the usual lexicographic ordering of place names, then the set of steps associated with marking 100, i.e., the set $S(100)$, is

$$\{< 100, T1, 1 >< 010, I1, 1 >, \ < 100, T1, 1 >< 010, I1, 2 >\}.$$

Similarly, $S(001) = \{< 001, T2, 1 >\}$. These two markings are the only stable reachable markings from the pictured initial marking. Since both $S(100)$ and $S(001)$ are finite, the activity network is stabilizing. Now consider the activity network of Figure 2.3. For this

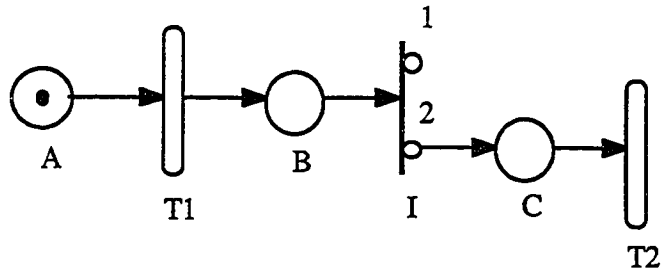


Figure 2.2: A Stablizing Activity Network

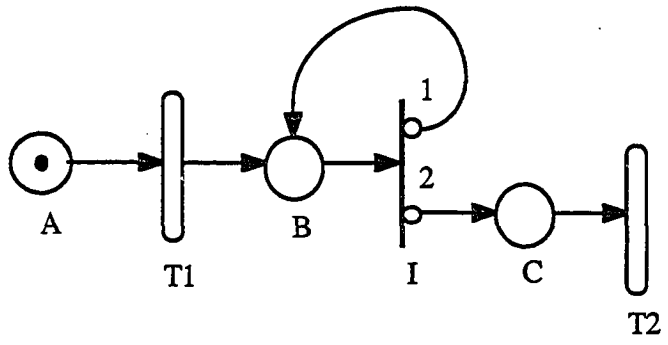


Figure 2.3: An Activity Network which is not Stablizing

network, $S(100)$ is the countably infinite set

$$\left\{ \begin{array}{l} < 100, T1, 1 >< 010, I1, 2 >, \\ < 100, T1, 1 >< 010, I1, 1 >, \\ < 100, T1, 1 >< 010, I1, 1 >< 010, I1, 1 >, \\ < 100, T1, 1 >< 010, I1, 1 >< 010, I1, 1 >< 010, I1, 1 >, \\ \vdots \end{array} \right\}.$$

Thus the activity network of Figure 2.3 is not stabilizing.

Generally, it is not decidable whether an activity network in a marking μ_0 is stabilizing. To see this, recall that it can be shown (see [76], for example) that extended Petri nets (Petri nets with inhibitor arcs) are equivalent, computationally, to Turing machines. The proof of this fact is by construction. Specifically, it can be shown that any register machine can be converted into an equivalent extended Petri net. In this case, the languages generated by the net can be taken to be the set of sequences of transitions that lead to a reachable marking. Given this equivalence, it is evident that activity networks are equivalent to Turing machines, since every extended Petri net is an activity network (transitions map to activities, places to places, and arcs to gates). In the context of an activity network, the language generated can be taken to be the set of steps with initial marking μ_0 (i.e., $S(\mu_0)$). Thus the class of languages generated by the set of possible activity networks is coextensive with the class of recursively enumerable sets. Since, generally, it can not be decided whether a recursively enumerable set is finite [37], we have the following theorem.

Theorem II.1 It is not decidable whether an activity network in a marking μ_0 is stabilizing.

There are, however, sufficient conditions by which the stabilizing property can be established, based on the structural properties and configuration of the instantaneous activities in the network. Identification of conditions is aided by the introduction of two properties of instantaneous activities. Specifically,

Definition II.9 An instantaneous activity is *self-disabling* if, given any reachable marking μ, it can only complete a finite number of times without any other activities completing.

This definition allows us to identify activities that will complete only a bounded number of times without a change in the marking of their input places caused by another activity. While this may be a difficult condition to check generally, it is easy to identify several frequently used activity-gate pairs that are self-disabling. For example, an activity with disjoint sets of input and output places and all default input gates (denoted by directed arcs) is self-disabling. In order to identify those activities which have no potentially unstabilizing interactions with other activities, we introduce the notion of a "cycle-free" instantaneous activity.

Definition II.10 An instantaneous activity I_1 is *cycle-free* if there does not exist a sequence of instantaneous activities $I_1, I_2, \ldots, I_n$ such that

$$\begin{aligned} &OP(I_1) \cap IP(I_2) \neq \emptyset \ \wedge \\ &OP(I_2) \cap IP(I_3) \neq \emptyset \ \wedge \\ &\quad \vdots \\ &OP(I_n) \cap IP(I_1) \neq \emptyset. \end{aligned}$$

Informally, then, an instantaneous activity is cycle-free if there does not exist a sequence of instantaneous activities that, through their completions, can affect the original activity's input places. Note that this is a purely structural condition and, even if an activity is not cycle-free, the enabling predicates of the activities in the cycle may be such that the cycle of completions can never occur. Furthermore, because the number of activities in a network is finite by definition, an algorithm exists to determine whether an activity is cycle-free. These two concepts provide us with criteria to identify sufficient conditions for an activity network to be stabilizing. Intuitively, if every instantaneous activity is cycle-free and self-disabling, then no instantaneous activity can complete an unbounded number of times without the intervening completion of a timed activity. More formally, we have the following theorem.

Theorem II.2 An activity network AN in a marking μ is stabilizing if every instantaneous activity of the network is cycle-free and self-disabling.

Proof:

The proof is by contradiction. Suppose there exists an activity network AN with activities A that is not stabilizing in a marking μ but where every instantaneous activity of the network is cycle-free and self-disabling. Then, by definition, since AN is not stabilizing there exists an activity which can complete an unbounded number of times without resulting in a stable marking. A self-disabling activity can only complete an unbounded number of times without reaching a stable marking if another instantaneous activity changes the marking of one of its input places. But every instantaneous activity in A is cycle-free, so this may not occur. Thus an activity network in a marking μ is stabilizing if every instantaneous activity in the network is cycle-free and self-disabling. □

For an example of an activity network that satisfies the conditions of the above theorem, see Figure 2.4. First, note that all the instantaneous activities in the network are self-disabling since they all have default input gates and disjoint input and output places. Secondly, note that all instantaneous activities are cycle free. Thus by Theorem II.2 this activity network is stabilizing. The stability of an activity network is an important necessary condition to insure that the probabilistic behavior of its stochastic extension is completely specified. Before the second necessary condition can be discussed, the definition of a stochastic activity network must be given.

Definition of a Stochastic Activity Network

Given an activity network which is stabilizing in some specified initial marking, a stochastic activity network is formed by adjoining functions C, F, and G, where C specifies the probability distribution of case selections, F represents the probability distribution functions of activity delay times, and G describes the sets of "reactivation markings" for each possible marking. Formally,

Definition II.11 A *stochastic activity network* (SAN) is a five-tuple

$$SAN = (AN, \mu_0, C, F, G)$$

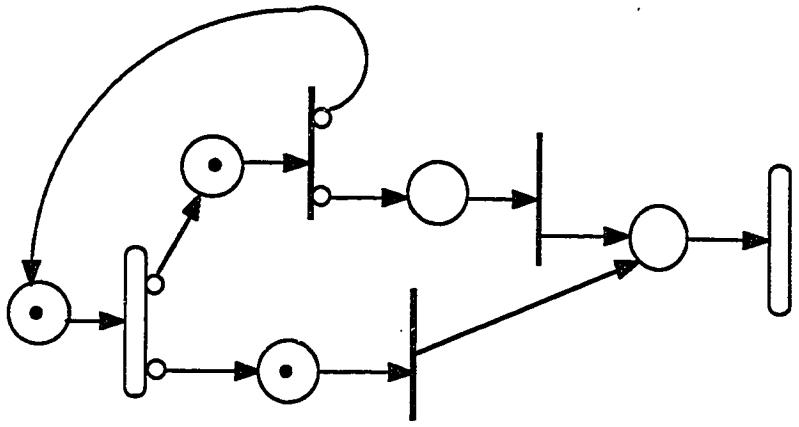


Figure 2.4: A Second Stabilizing Activity Network

where:

1. $AN = (P, A, I, O, \gamma, \tau, \iota, o)$ is an activity network.
2. $\mu_0 \in M_P$ is the *initial marking* and is a stable marking in which AN is stabilizing.
3. C is the *case distribution assignment*, an assignment of functions to activities such that for any activity a, $C_a : M_{IP(a) \cup OP(a)} \times \{1, \ldots, \gamma(a)\} \rightarrow [0, 1]$. Furthermore, for any activity a and marking $\mu \in M_{IP(a) \cup OP(a)}$ in which a is enabled, $C_a(\mu, \cdot)$ is a probability distribution called the *case distribution of a in μ*.
4. F is the *activity time distribution function assignment*, an assignment of continuous functions to timed activities such that for any timed activity a, $F_a : M_P \times \mathbb{R} \rightarrow [0, 1]$. Furthermore, for any stable marking $\mu \in M_P$ and timed activity a which is enabled in μ, $F_a(\mu, \cdot)$ is a continuous probability distribution function called the *activity time distribution function of a in μ*; $F_a(\mu, \tau) = 0$ if $\tau \leq 0$.
5. G is the *reactivation function assignment*, an assignment of functions to timed activities such that for any timed activity a, $G_a : M_P \rightarrow \wp(M_P)$, where $\wp(M_P)$ denotes

the power set of M_P. Furthermore, for any stable marking $\mu \in M_P$ and timed activity a which is enabled in μ, $G_a(\mu, \cdot)$ is a set of markings called the *reactivation markings* of a in μ.

Recall that an activity is enabled if all of its input gates hold. While this concept is suffices to describe the behavior of an activity network, several additional terms are needed to describe the behavior of a stochastic activity network. Figure 2.5 aids in the description of these terms. In particular, since timed activities represent operations in a modeled system, events must be defined to denote the start and finish of these operations. The start of an operation is signaled by an *activation* of an activity. An activation of an activity will occur if 1) the activity becomes enabled (illustrated by the first time-line) or 2) the activity completes and is still enabled (illustrated by the second time-line). Some time after an activity is activated it will either *complete* (illustrated by the first time-line) or be *aborted* (illustrated by the third time-line). The activity will complete if it remains enabled throughout its activity time (to be defined momentarily); otherwise it is aborted.

The activity time distribution function specifies (probabilistically) the *activity time* of an activity, i.e., the time between its activation and *completion.* Any continuous distribution (e.g., exponential, normal) is a legal activity time distribution, although the choice of distribution will affect the applicability of many solution methods. Both the distribution type and its parameters can depend on the global marking of the network at the activation time of the activity. Activity times are assumed to be mutually independent random variables.

Two other functions are associated with an activity network to form a SAN. In particular, the case distribution specifies (probabilistically) which case is to be chosen upon the completion of an activity. These probabilities can depend on the markings of the input and output places of the activity at its completion time. A reactivation function is also associated with each timed activity. This function specifies, for each marking, a set of *reactivation markings.* Given that an activity is activated in a specific marking, it is reactivated (i.e., activated again) whenever any marking in the set of reactivation markings for the activation marking is reached before it completes (as illustrated by the fourth time-line). Probabilistically, the reactivation of an activity is exactly the same as

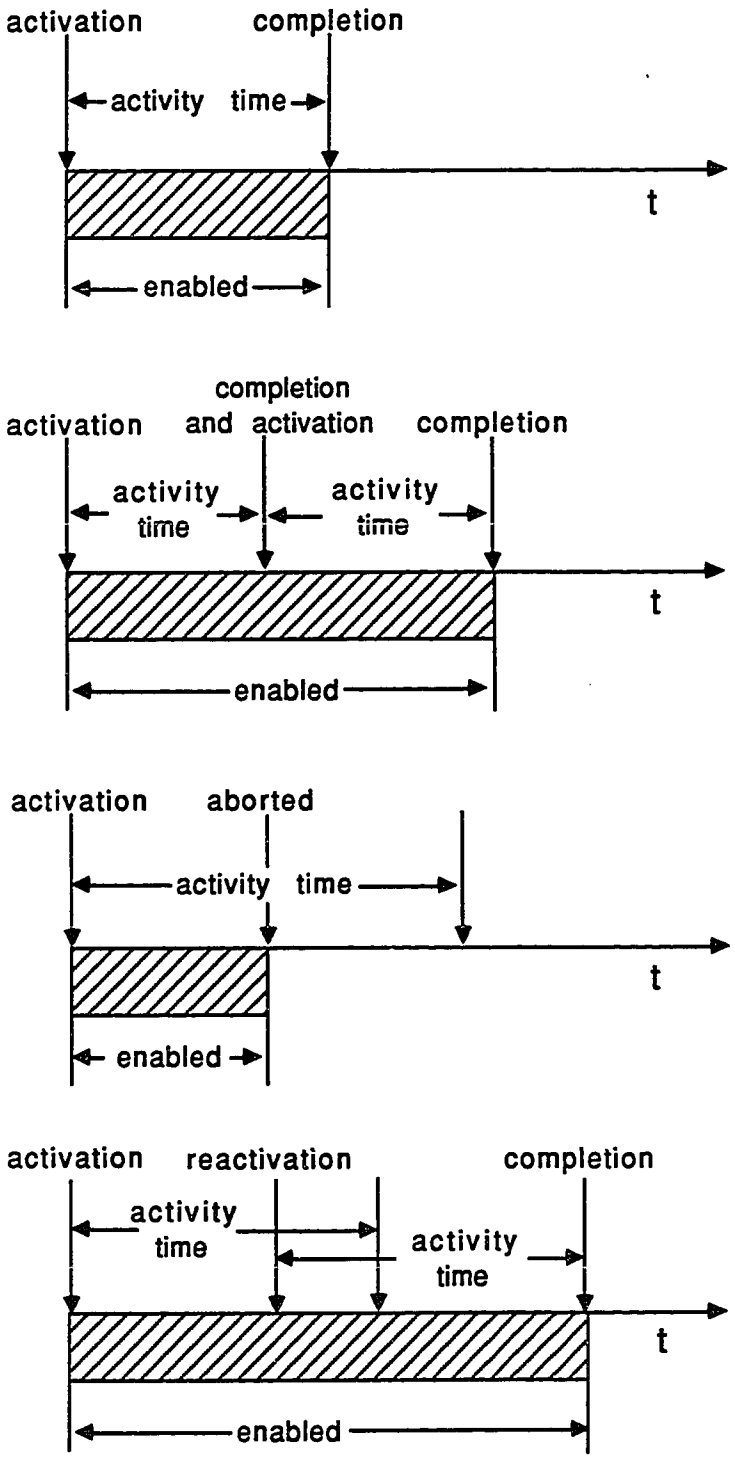


Figure 2.5: Terms Related to the Execution of a Timed Activity

an activation; a new activity time distribution is selected based on the current marking. This provides a mechanism for restarting activities that have been activated, either with the same or different distribution. This decision is made on a per activity basis (based on the reactivation function) and, hence, is not a net-wide execution policy.

The definition of a stochastic activity network presented here differs from that presented earlier in [66] in three respects. First, it is required that probabilities associated with cases depend only on input and output places of the associated activity. This requirement permits the development of more efficient algorithms to test whether the probabilistic behavior of a stochastic activity network is completely specified. Since any place in a network can be made to be an input or output place of any activity, this is not a restrictive requirement.

Second, the new definition requires that the probability distribution function associated with each timed activity in a possible marking be continuous. This requirement is to insure that no two timed activities complete at the same time, since the behavior in this instance is not specified. This requirement is not overly strict for stochastic activity networks that are to be solved analytically, since the solution method usually places stricter constraints on the distributions. In the case of SANs that are to be solved via simulation, the ambiguity can be avoided by assigning an ordering to timed activities that may complete at the same time.

Third, the current definition does not require that the underlying activity network be "well behaved" [66] in its initial marking. An activity network is said to be *well behaved* in an initial marking μ if

1. No infinite sequence of instantaneous activities may complete in a marking reachable from marking μ, and

2. If a marking reachable from μ has more than one enabled instantaneous activity, then, from that marking, all possible sequences of reachable markings result in the same stable marking.

The requirement that the underlying activity network be well behaved is more strict than the requirement that the underlying AN be stabilizing, and does lead to stochastic activity networks whose probabilistic behavior is completely specified. However, delaying the introduction of conditions of this type until after the stochastic extension is defined allows us to identify a larger class of networks whose probabilistic behavior is completely specified.

Stochastic Activity Network Behavior

Before discussing in detail how activity time distributions, case distributions, and reactivation functions determine an activity network's behavior, it helps to describe, informally, how a network executes in time. In particular, one can think of an execution of a SAN as a sequence of configurations, the interpretation being that for each configuration $< \mu, a, c >$ the SAN was in marking μ, activity a completed, and case c was chosen. In any marking μ, the activity that completes is selected from the set of *active* activities in μ, i.e., the set of those activities which have been activated but have not yet completed or aborted. After each activity completion and case selection, the set of activities which are active is altered as follows. If the marking reached (as specified by the yields relation) is stable, then

1. the activity which completed is removed from the set of active activities,
2. activities which are no longer enabled in the reached marking are removed from the set of active activities,
3. activities for which the reached marking is a reactivation marking are removed from the set of active activities, and
4. activities which are enabled but not in the set of active activities are placed in the set (including those which were reactivated).

In contrast, if the marking reached is not stable, then timed activities (other than the one that just completed, if timed) are not added or deleted from the set. Instead,

1. the activity which completed is removed,
2. instantaneous activities which are no longer enabled in the reached marking are removed, and
3. instantaneous activities which are enabled but not in the set are added.

The choice of activity to complete from the set of active activities is determined by the activity time distribution function of each activity in the set and the relative priority of timed and instantaneous activities (as specified by the definition of "may complete"). If there are one or more instantaneous activities in the set, one of them is chosen (non-deterministically) to complete. If not, the timed activity with the earliest completion time is selected (stochastically) based on the activity times of the set of active activities. Recall that the activity time distribution function defines the time between an activity's activation and completion. After the activity to complete is selected, a case of the activity is chosen based on its case distribution in the current marking, and a new marking is reached. The set of active activities is "initialized" at the start of an execution by adding all those activities to the set that are enabled in the initial marking. Note that, because choices between active instantaneous activities are made non-deterministically but not probabilistically, there may be networks for which these choices lead to behaviors which are not probabilistically specified. We now investigate conditions under which probabilistic behavior of the network is completely specified.

Well Specified Stochastic Activity Networks

Two types of nondeterminism can occur in stochastic activity networks: 1) uncertainty as to which activity completes among the active activities, and 2) uncertainty as to which case is chosen of the activity that completes. To aid in the discussion that follows, we will refer to a choice of the first type as an *activity choice* and a choice of the second type as a *case choice*.

In stochastic activity networks, case choices are quantified by the assignment of a case distribution to each activity. Activity choices are quantified partially by the assignment of an activity time distribution to each timed activity. However, the activity time distribution does not completely quantify this type of non-determinism, since the behavior is not defined in the case that two activities have the same completion time. This will never occur for two timed activities, since all activity time distributions are continuous. It will occur, however, whenever two or more instantaneous activities are enabled, since instantaneous activities complete in zero time. In this situation, the choice of which

activity completes next is non-deterministic and not quantified by either the activity time distribution function assignment or the case distribution assignment.

Since we are interested in possible sequences of timed activity completions and reached stable markings, we would like the probability distribution on the choice of next stable marking to be the same regardless of any non-probabilistically quantified activity choices that have been made. To investigate this more formally, we introduce the notion of a "stable step".

Definition II.12 Let $S = (AN, \mu_0, C, F, G)$ be a stochastic activity network and s be a step of AN with initial marking μ_0, then s is a *stable step* if the resulting marking of s is stable.

Stable steps can be thought of as "jumps" in the execution of a stochastic activity network that take the network from stable marking to stable marking via the completion of a timed activity and zero or more instantaneous activities. There may be several steps with the same first marking and activity, but a different final marking. Accordingly, we define the "set of next stable markings" for a stable marking upon completion of an activity a as follows.

Definition II.13 Let $S = (AN, \mu_0, C, F, G)$ be a stochastic activity network and $\mu \in SR(AN, \mu_0)$. The set of *next stable markings* for S in μ upon completion of a is the set

$$NS(\mu, a) = \left\{ \mu' \;\middle|\; \begin{array}{l} \exists \text{ a stable step } s \text{ from } \mu \text{ to } \mu' \text{ such that} \\ a \text{ is the activity of the first configuration of } s \end{array} \right\}.$$

This set allows us to focus our attention on those stable markings that may be reached from a particular stable marking by completion of a specific timed activity. In order to insure that the probability distribution over next stable markings is invariant over activity choices between instantaneous activities, we must define a measure on stable steps that captures the probability that a stable step is taken given that a set of activity choices is made in a manner such that the step is possible. The case construct allows us to define this probability. Specifically,

Definition II.14 Let $S = (AN, \mu_0, C, F, G)$ be a SAN and $s = < \mu^1, a^1, c^1 >< \mu^2, a^2, c^2 > \cdots < \mu^n, a^n, c^n >$ be a path of S. Then, *the probability of* s, denoted $Pr(s)$, is

$$Pr(s) = C_{a^1}(\mu^1_{IP(a^1)\cup OP(a^1)}, c^1) \times C_{a^2}(\mu^2_{IP(a^2)\cup OP(a^2)}, c^2) \ldots \times C_{a^n}(\mu^n_{IP(a^n)\cup OP(a^n)}, c^n)$$

where $\times$ is taken to be normal multiplication on the set of real numbers.

This function defines the probability that a stable step is taken given that a set of activity choices was made such that the step may occur. Since we want to insure that the probability distribution over next stable markings upon completion of a timed activity is invariant over possible sets of activity choices, we consider the set of steps from some stable marking μ to a stable marking $\mu' \in NS(\mu, a)$, by completion of a timed activity a which may complete in μ. Formally, let

$$P^a_{\mu,\mu'} = \{ s \mid s \text{ is a stable step from } \mu \text{ to } \mu' \text{ with first activity } a \}.$$

Different stable steps in this set can arise from different sets of activity choices. In order to specify the set of stable steps from one marking to another upon completion of some timed activity for a single set of activity choices, it helps to define a relation relating stable steps that can occur under a single set of activity choices. Specifically, define the relation R on $P^a_{\mu,\mu'}$ to be

$$R = \left\{ (s, s') \;\middle|\; \begin{array}{l} \text{for every configuration } < \mu, a, c > \text{ in } s \text{ and configuration} \\ < \mu', a', c' > \text{ in } s' \text{ such that } \mu = \mu',\ a = a' \end{array} \right\}.$$

Thus two stable steps are related if, for every marking they share in common, the same activity choice is made. Note that while R is not an equivalence relation, it is a compatibility relation (i.e., it is reflexive and symmetric). While a compatibility relation does not necessarily define a partition of a set, it does define a covering of a set, by the *maximal compatibility classes* of the relation. Recall that (as in [87]) a subset $C \subseteq P^a_{\mu,\mu'}$ is called a *maximal compatibility class* if any every element of C is related to every other element of C and no element of $P^a_{\mu,\mu'} - C$ is related to all the elements of C. Each maximal compatibility class contains stable steps that correspond to a single set of activity choices.

More specifically, note that all stable steps in C correspond to steps from μ to μ' by completion of timed activity a under some set of activity choices. The probability that μ' is reached from μ by completion of a, given that a particular set of activity choices has been made, is thus the sum of the probabilities of all stable steps in C, i.e.,

$$P(C) = \sum_{s \in C} Pr(s).$$

All activity choices within stable steps correspond to choices between active instantaneous activities and, hence, are not probabilistically specified. Therefore, for a stochastic activity network to be completely probabilistically specified, $P(C)$ must be the same for all maximal compatibility classes C. To express this precisely, we introduce the notion of a "well specified stochastic activity network".

Definition II.15 A stochastic activity network $S = (AN, \mu_0, C, F, G)$ is *well specified* if, for every marking $\mu \in SR(AN, \mu_0)$, each activity a which may complete in μ, and all $\mu' \in NS(\mu, a)$, P(C) is identical for all maximal compatibility classes C of R defined on $P^a_{\mu,\mu'}$.

The above definition identifies a class of networks whose behavior is completely specified, probabilistically, with respect to all notions of state that we intend to consider.

It is interesting to compare the notion just presented to the well behaved notion used previously. In particular, one can show that every activity network well behaved in some marking μ_0 is well specified for any choice of activity time distributions, reactivation functions, and case probabilities. We state this fact as a theorem.

Theorem II.3 If an activity network is well behaved in a marking μ_0, then it is well specified for any choice of activity time distributions, reactivation functions, and case distributions.

Proof:

Suppose an activity network AN is well behaved in a marking μ_0. Then, for every marking μ reachable from μ_0, no infinite sequence of activities may complete in μ. Thus AN is stablizing in μ_0. Augment AN with arbitrary activity distributions, reactivation functions,

case distributions, and initial marking μ_0 to form a SAN. Now, recall that this SAN is well specified if for every stable marking μ reachable from μ_0, each activity a which may complete in μ, and each $\mu' \in NS(\mu, a)$, P(C) is identical for all maximal compatibility classes C or R defined on $P^a_{\mu,\mu'}$. Consider an arbitrary stable marking μ reachable from μ and activity a which may complete in μ. Since S is well behaved, one of three situations may arise upon completion of a in μ.

In the first situation, all markings in the set of next possible markings are stable. There is thus only one maximal compatibility class for this marking and activity and the criterion of Definition II.15 is satisfied.

In the second situation, at least one marking in the set of next possible markings is unstable, and all possible unstable markings that may be reached before reaching another stable marking have at most one instantaneous activity enabled. Then, as in the first situation, there is only one maximal compatibility class for this marking and activity and, therefore, the criterion of Definition II.15 is satisfied.

In the final situation, at least one marking in the set of next possible markings is unstable, and some unstable marking that may be reached before reaching another stable marking has two or more instantaneous activities enabled. But, since S is well behaved, all possible sequences of reachable markings, from that marking, result in the same next stable marking. Thus while there may be more than one compatibility class for the marking and activity, they all result in the same single marking with probability one, and hence, $P(C)$ is identical for all maximal compatibility classes. Again the criterion of Definition II.15 is satisfied for this marking and activity.

Since the criterion of the definition is satisfied for each possible situation for every reachable marking and activity that may complete in the marking, the SAN is well specified, for any choice of activity time distributions, reactivations, and case distributions. □

It should be noted, however, that the converse of II.3 is not a theorem and, hence, well-specified SANs are properly more general than well-behaved SANs. To see this, consider the stochastic activity network of Figure 2.6. This SAN is well specified in the pictured marking, since the activity choice which is made after completion of the enabled timed

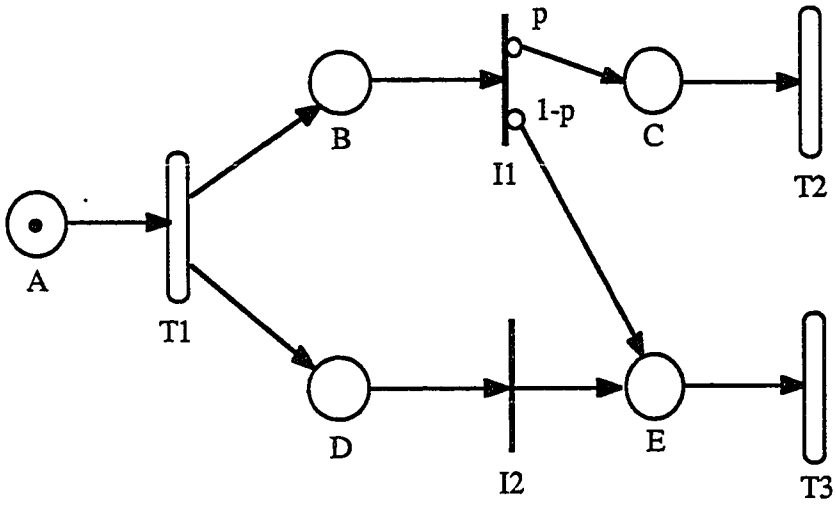


Figure 2.6: A Well Specified, but not Well Behaved, Stochastic Activity Network

activity does not affect the distribution of next stable markings. It is not well behaved though, since two instantaneous activities may be enabled and more than one stable marking can be reached from the current marking. Recall that in order for an activity network to be well behaved, whenever a reached marking has two or more instantaneous activities enabled, all possible sequences of reachable markings must result in the same stable marking.

Any algorithm to determine whether a given stochastic activity network is well specified must check that the probability distribution over each next stable markings does not depend on the set of activity choices that is made. This condition can be checked using techniques developed to find the set of all maximal compatibles [46]. The following algorithm checks this for a particular stable marking and timed activity.

Algorithm II.1 (Determines whether the next stable marking probability distribution is invariant over possible sets of activity choices for a stable marking μ and activity a which can complete in μ, and if so, computes this distribution.)

> Compute the set of all stable steps where the initial marking is μ and timed activity is a.
>
> Group the set of stable steps computed above according to the resulting marking of each step. Denote the subset containing the stable steps from μ to μ' by completion of timed activity a as $P^a_{\mu,\mu'}$.
>
> For each subset $P^a_{\mu,\mu'}$:
>
> > Construct the set of maximal compatibles of R on $P^a_{\mu,\mu'}$.
> >
> > Compute $P(C)$ for each maximal compatible C.
> >
> > If $P(C)$ is not identical for all compatibles C then
> >
> > > Signal SAN is not well specified and abort algorithm.
>
> Next $P^a_{\mu,\mu'}$.
>
> Return next stable marking probability distribution for marking and activity.

For convenience in that which follows, we denote this distribution by the function $h_{\mu,a} : NS(\mu,a) \rightarrow [0,1]$, where for $\mu' \in NS(\mu,a)$, $h_{\mu,a}(\mu')$ is the probability that μ' is reached given that the SAN is in μ and a completes.

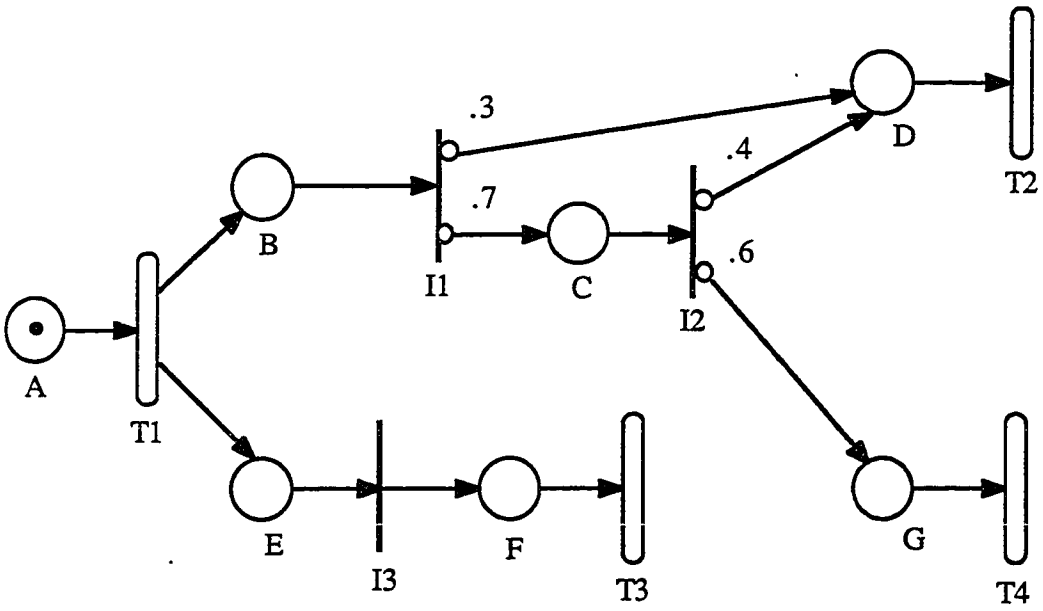


Figure 2.7: A Well Specified Stochastic Activity Network

The following example illustrates the use of Algorithm II.1. Specifically, consider the stochastic activity network of Figure 2.7. Here the case distribution for each activity is denoted by the number next to each case for the activity. In addition, markings are denoted as vectors, assuming the usual lexicographic ordering of place names. With these facts in mind, we will show (using Algorithm II.1) that the next stable marking probability distribution is invariant over possible sets of activity choices for the initial marking 1000000 (lexicographic ordering on place names) and timed activity $T1$. As per the algorithm, the set of all stable steps is computed first, and is shown in Figure 2.8. These steps are then used to determine the set of possible next stable markings ($NS(1000000, T1)$), which is found to be $\{0001010, 0000011\}$. The set of stable steps is then split into two subsets, according to their resulting marking. These two subsets serve as input to the portion of the algorithm that computes sets of maximal compatibles, checks to see that the probability measure is invariant over all possible compatibles, and computes the next stable marking probabilities. As per the algorithm, the set of maximal compatibles corresponding to the

$$\left\{\begin{array}{l}
< 1000000, T1, 1 >< 0100100, I1, 1 >< 0001100, I3, 1 >, \\
< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I2, 1 >< 0001100, I3, 1 >, \\
< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I2, 2 >< 0000101, I3, 1 >, \\
< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I3, 1 >< 0010010, I2, 1 >, \\
< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I3, 1 >< 0010010, I2, 2 >, \\
< 1000000, T1, 1 >< 0100100, I3, 1 >< 0100010, I1, 1 >, \\
< 1000000, T1, 1 >< 0100100, I3, 1 >< 0100010, I1, 2 >< 0010010, I2, 1 >, \\
< 1000000, T1, 1 >< 0100100, I3, 1 >< 0100010, I1, 2 >< 0010010, I2, 2 >
\end{array}\right\}$$

Figure 2.8: Set of Stable Steps for Stochastic Activity Network of Figure 2.7 from Marking 10000000 by Completion of Activity $T1$.

resulting marking 0001010 is then computed, and found to consist of the three elements

$C_1 = \{< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I2, 1 >< 0001100, I3, 1 >,$
$< 1000000, T1, 1 >< 0100100, I1, 1 >< 0001100, I3, 1 >\},$

$C_2 = \{< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I3, 1 >< 0010010, I2, 1 >,$
$< 1000000, T1, 1 >< 0100100, I1, 1 >< 0001100, I3, 1 >\},$ and

$C_3 = \{< 1000000, T1, 1 >< 0100100, I3, 1 >< 0100010, I1, 2 >< 0010010, I2, 1 >,$
$< 1000000, T1, 1 >< 0100100, I3, 1 >< 0100010, I1, 1 >\}.$

P(C) is then computed for each maximal compatible C, and $P(C_1) = P(C_2) = P(C_3) =$ .58. Similarly, computing the set of maximal compatibles of stable steps with resulting

marking 0000011, we obtain:

$$C_1 = \{< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I2, 2 >< 0000101, I3, 1 >\}$$

$$C_2 = \{< 1000000, T1, 1 >< 0100100, I1, 2 >< 0010100, I3, 1 >< 0010010, I2, 2 >\}$$

$$C_3 = \{< 1000000, T1, 1 >< 0100100, I3, 1 >< 0100010, I1, 2 >< 0010010, I2, 2 >\}$$

For these compatibles, $P(C_1) = P(C_2) = P(C_3) = .42$. Since the probability measure is the same for each maximal compatible in a set, for both sets, the next stable marking probability distribution is invariant over possible set of activity choices for this SAN, starting marking and timed activity.

By definition, then, a stochastic activity network is well specified if this distribution is invariant for all stable reachable markings and activities which may complete in these markings. In cases where the set of stable reachable markings is finite, this fact leads immediately to an algorithm to check whether a SAN is well specified. Specifically, consider the following algorithm.

Algorithm II.2 (Determines whether a SAN with a finite reachability set is well specified and computes next stable marking probability distributions)

For each $\mu \in SR(AN, \mu_0)$ and activity a which may complete in μ:

Determine whether the next stable marking probability distribution is invariant over possible sets of activity choices for this choice of μ and a.

If distribution is not invariant for this μ and a then

Signal SAN is not well specified and abort algorithm.

Next $\mu \in SR(AN, \mu_0)$ and activity a which may complete in μ.

Signal stochastic activity network is well specified.

Note that this algorithm is simply an iterative application of Algorithm II.1. While Algorithm II.1 suffices to determine whether the next stable marking probability distribution is invariant for a particular marking μ and activity a, its performance can be improved upon by using information concerning the structure of the network. This technique makes use of the concept of "dependent instantaneous activities". Specifically,

Definition II.16 Let I_1 and I_2 be instantaneous activities of some activity network. Then I_1 and I_2 are *dependent* if

$$(IP(I_1) \cup OP(I_1)) \cap (IP(I_2) \cup OP(I_2)) \neq \emptyset.$$

Informally, then, two instantaneous activities are dependent if they have common input or output places. Two activities which are dependent can affect each other by changing the marking of each other's input or output places. In order to identify instantaneous activities that can affect each other through a sequence of completions we look at the transitive closure of a relation based on the above definition. Specifically, let DEP denote a relation on the set of instantaneous activities of a SAN such that I_1 DEP I_2 if I_1 and I_2 are dependent. Furthermore, let $\overset{*}{DEP}$ denote the transitive closure of DEP. It is easy to see that $\overset{*}{DEP}$ is an equivalence relation; thus it partitions the set of instantaneous activities. The blocks of the partition are sets of activities whose order of completion may affect the probability distribution of next stable markings. On the other hand, pairs of activities from different blocks cannot affect each other by completing (since activities can only change the marking of their input or output places and case probabilities depend only on input and output places). This suggests that steps within each subnetwork defined by activities in each block can be considered individually and combined to determine the total probability for a possible next stable marking.

In order to explore this idea in more detail, we define the notion of an instantaneous subnetwork of a SAN constructed from a set of activities.

Definition II.17 Given a stochastic activity network $S = (AN, \mu_0, C, F, G)$, underlying activity network $AN = (P, A, I, O, \gamma, \tau, \iota, o)$, and set of instantaneous activities $A' \subseteq A$ the *instantaneous subnetwork of S with respect to A'* is a structure (M', μ_0', C') where

1. $M' = (P', A', I', O', \gamma', \tau', \iota', o')$ is an activity network with
   (a) $P' = \{p \mid p \in P \text{ and } p \in IP(a) \cup OP(a) \text{ for some } a \in A'\}$,
   (b) A' is some specified set of instantaneous activities,
   (c) $I' = \{g \mid g \in I \text{ and } g \in \iota^{-1}(a) \text{ for some } a \in A'\}$,
   (d) $O' = \{g \mid g \in O \text{ and } g \in o^{-1}(a, c) \text{ for some } a \in A' \text{ and } c = 1, 2, \ldots, \gamma(a)\}$, and

(e) γ', τ', ι', and o' are the functions γ, τ, ι, and o, respectively, restricted to P', A', I', and O'.

2. $\mu'_0 = \mu_0$ restricted to places P', and

3. C' is the function C restricted to A'.

While (M', mu'_0, C') does not fit the definition of a stochastic activity network precisely since the initial marking is not stable, it does provide us with a network made up of instantaneous activities where all the case probabilities are specified. The revised algorithm presumes that such a subnetwork is constructed for each set of activities corresponding to a block of the partition defined by $\overset{*}{DEP}$. By the nature of $\overset{*}{DEP}$, these subnetworks do not interact with one another. This fact is exploited in the revised algorithm presented below.

Unlike Algorithm II.1, which computes the set of all stable steps for the given marking and activity immediately, the revised algorithm accomplishes the same goal in two smaller steps. First, it computes the set of "next possible markings" for the given starting marking and activity. The set of next possible markings is the set of markings that can be reached by one application of the yields relation, i.e. for a given marking μ and activity a

$$NP(\mu, a) = \{\mu' \mid \mu \overset{a,c}{\rightarrow} \mu' \text{ for some } a \in A \text{ and } c \in \{1, \ldots, \gamma(a)\}\}.$$

Each of these markings can be either stable or unstable. If a marking is stable, then it is a next stable marking for the specified starting marking and activity. Furthermore, its probability of occurrence is just the sum of the probabilities of all cases that lead to that marking. Since only probabilistically specified activity choices were made in reaching the marking, the network is well specified. A more complicated situation exists for each unstable marking in $NP(\mu, a)$.

These markings are unstable markings that can be reached after one application of the yields relation and, hence, situations where one or more instantaneous activities must be completed to find possible resulting next stable markings. To find these, consider the instantaneous subnetworks previously constructed. First, note that although the sets of places defined by each subnetwork are disjoint, they may not partition the set of places of the entire network, since there may be places that are connected only to timed activities.

The marking of these places will not change by completion of instantaneous activities and hence will remain the same in all next stable markings of the network reached from this marking. In the algorithm which follows, the marking of the places connected only to timed activities is denoted as μ'_s, for each $\mu' \in NP(\mu, a)$. Similarly, the initial (unstable) markings of each of the n subnetworks are denoted as $\mu'_1, \mu'_2, \ldots, \mu'_n$, respectively, for each $\mu' \in NP(\mu, a)$. Now, since the markings of places of different subnetworks are independent of each other, sets of next stable markings can be computed for each subnetwork independently and be combined to obtain "global" next stable markings for the entire network.

Computation of the local next stable markings, and the subsequent check that the probabilities these markings are invariant over possible sets of activity choices for each subnetwork, is done in a manner similar to that of Algorithm II.1, except that the initial marking of each of the possible paths to a stable marking is not itself stable. Since these paths are suffixes of stable steps, we call them "partial stable steps". A *partial stable step* is a stable step without the initial configuration. Except for this difference, the invariant check and computation of probabilities is done exactly as in Algorithm II.1. In the algorithm presented below, for each subnetwork i, the set of (subnetwork) next stable markings is denoted as NS_i and the probability that subnetwork marking $\mu''_{i,j}$ is reached from subnetwork marking μ'_i is denoted as $\hat{h}_{\mu'_i}(\mu''_{i,j})$.

After the possible next stable markings and probabilities of these markings have been computed for each subnetwork, they are combined to construct next stable markings and probabilities for the entire network. Possible next stable markings for the entire network are constructed by forming all possible combinations of subnetwork next stable markings together with μ'_s. Each marking constructed this way is denoted as the concatenation of its constituent subnetwork markings together with μ'_s. The probability of each of these global markings is then computed as the product of the probabilities of each of the constituent markings. Since each global marking could also be reached in other ways (i.e., from another $\mu' \in NP(\mu', a)$), the computed probability obtained in each way is summed to obtain the total probability for this next stable marking.

A more precise description of the algorithm is the following.

Algorithm II.3 (Uses concept of instantaneous subnetworks to determine, given a stable marking μ and activity a which can complete in μ, whether the next stable marking probability distribution is invariant over possible sets of activity choices, and if so, computes this distribution.)

Let $NS(\mu, a)$ equal the null set.

Let $h_{\mu,a} = 0$.

Compute $NP(\mu, a)$.

For each $\mu' \in NP(\mu, a)$:

Let $\bar{h}_{\mu,a}(\mu') = \sum_{c \text{ such that } \mu \xrightarrow{a,c} \mu'} C_a(\mu_{IP(a) \cup OP(a)}, c)$.

If μ' is stable then

Add μ' to $NS(\mu, a)$.

Let $h_{\mu,a}(\mu') = \bar{h}_{\mu,a}(\mu')$.

else

For each instantaneous subnetwork i, $i = 1$ to n:

Restrict μ' to places of the subnetwork (denote this μ'_i).

Compute the set of all partial stable steps of the subnetwork with initial marking μ'_i.

Compute the set of resulting stable markings from the set of partial stable steps. Denote these k_i markings $\mu''_{i,1}, \mu''_{i,2}, \ldots, \mu''_{i,k_i}$.

Group the set of partial stable steps computed above according to their resulting marking. Denote the subset containing partial stable steps from μ'_i to $\mu''_{i,j}$ as $P_{\mu'_i,\mu''_{i,j}}$.

For each $P_{\mu'_i,\mu''_{i,j}}$, $j = 1$ to k_i:

Construct the set of maximal compatibles of R on $P_{\mu'_i,\mu''_{i,j}}$.

Compute $P(C)$ for each maximal compatible C.

If $P(C)$ is not identical for all compatibles C then

Signal SAN is not well specified and abort algorithm.

else

Add $\mu''_{i,j}$ to $NS_i(\mu'_i)$.

Let $\hat{h}_{\mu'_i}(\mu''_{i,j}) = P(C)$.

Next j.

Next i.

{* Now form global next stable marking from subnetwork results *}

For $j_1 = 1$ to k_1

For $j_2 = j_1$ to k_2

$\ddots$

For $j_n = j_{n-1}$ to k_n

Add $\mu'_{1,j_1}\mu'_{2,j_2}\cdots\mu'_{n,j_n}\mu'_s$ to $NS(\mu, a)$.

Let $h_{\mu,a}(\mu'_{1,j_1}\mu'_{2,j_2}\cdots\mu'_{n,j_n}\mu'_s) =$
$h_{\mu,a}(\mu'_{1,j_1}\mu'_{2,j_2}\cdots\mu'_{n,j_n}\mu'_s) + \bar{h}_{\mu,a}(\mu')\prod_{i=1 \text{ to } n}\hat{h}_{\mu'_i}(\mu'_{i,j_i})$.

Next j_n.

Next j_2.

Next j_1

Next $\mu' \in NP(\mu, a)$.

Return next stable marking probability distribution for marking and activity.

This algorithm has significant advantages over Algorithm II.1 for systems that have several instantaneous subnetworks, and reduces to Algorithm II.1 when there is a single subnetwork. To illustrate this, consider again the stochastic activity network of Figure 2.7, which was shown to be well specified using Algorithm II.1. Figure 2.9 depicts this activity network with the instantaneous subnetworks outlined. As shown by Algorithm II.3, each of these subnetworks can be analyzed separately to check that the network is well specified and to compute the next stable state probability distribution. In order to

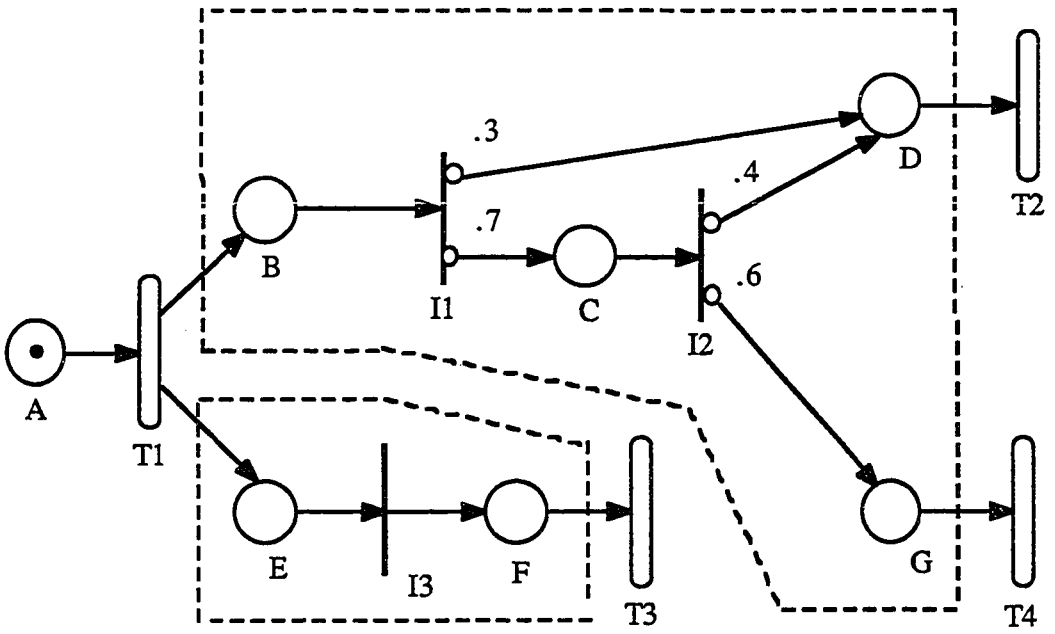


Figure 2.9: A Well Specified Stochastic Activity Network With Instantaneous Subnetworks Noted

do this, we must first compute the set of next possible markings, which in this case is {0100100}. The probability this marking, denoted $\bar{h}_{1000000,T1}(0100100)$, is 1. Now, since the marking is not stable, the set of next stable states and their probabilities must be computed for each instantaneous subnetwork. For the first subnetwork, which contains $I1$ and $I2$, the set of partial stable steps is

$$\left\{ \begin{array}{l} < 1000, I1, 1 >, \\ < 1000, I1, 2 >< 0100, I2, 1 >, \\ < 1000, I1, 2 >< 0100, I2, 2 > \end{array} \right\}$$

and set of possible next stable markings, NS_1 is $\{0010, 0001\}$. There is one maximal compatible corresponding to each possible next stable marking. The first,

$$C = \{< 1000, I1, 1 >, < 1000, I1, 2 >, < 0100, I2, 1 >\}$$

has probability $P(C) = .58$ and, hence, $\hat{h}_{0010} = .58$. The second, corresponding to the stable marking 0001, is

$$C = \{< 1000, I1, 2 >< 0100, I2, 2 >\}$$

and has probability $\hat{h}_{1000}(0001) = .42$. The computations for the second subnetwork are even simpler. For this subnetwork, the set of partial stable steps is the singleton set

$$\{< 10, I3, 1 >\}$$

where the set of possible stable markings, NS_2 is $\{01\}$ and $\hat{h}_{10}(01) = 1$.

Since the next stable state probabilities are invariant for each subnetwork, they are invariant for the entire network. The global next stable marking probabilities are computed by forming possible combinations of the local next stable markings. When this is done, the set of next stable markings is found to be $NS(1000000) = \{0001010, 0000011\}$ with proabilities $h_{1000000,T1}(0001010) = .58$ and $h_{1000000,T1}(0000011) = .42$. This result matches that computed previously using Algorithm II.1.

CHAPTER III

PERFORMANCE VARIABLES

Introduction

The previous chapter presented a general framework for the representation of distributed systems using stochastic activity networks. These networks have properties that allow them to be solved either by analysis or simulation, depending on specific model characteristics. However, before methods to do this can be developed, it is necessary to specify the range of questions (i.e. "types" of performance variables) that will be considered. To some extent, this range is determined by the choice of representation scheme. For example, if queueing networks are employed, one is typically limited to asking questions regarding server utilizations, queue lengths, waiting times, and service times. If stochastic activity networks are used, the class is larger, due to the lower-level nature of the model primitives. On the other hand, as with queueing networks, the spectrum of variables that are considered directly affects the techniques that may be used to obtain a solution.

It is therefore useful to formally categorize performance variables in a manner that suggests methods which may be used to obtain their solution. Previous work done regarding "reward models" [38] (and associated "reward variables") provides an instructive example in this regard. Informally, a reward model consists of a stochastic process and a "reward structure". The reward structure relates possible behaviors of the process to a specified performance variable. Typically, this is done by associating a "reward rate" with each state, the interpretation being that this rate is the rate at which reward accumulates

while the process is in the state. The performance variable in this case is then taken to be the reward accumulated over some utilization interval (either finite or infinite). By associating different reward rates to states, one can construct performance variables with many different interpretations.

We take a similar approach in this chapter, but develop reward structures that quantify behaviors at the stochastic activity network level, instead of the state level. This approach has several distinct advantages over the state-level approach outlined above. First, the assignment of rewards and interpretation of solutions is more natural, since it is done at the level at which the modeler thinks. Second, since rewards are assigned at the network level, they can be used in the construction procedure (i.e. the procedure by which a base model or simulation program is generated from the network representation and performance variable).

The remainder of this chapter is organized as follows. In the following section, traditional reward models and variables are reviewed and a general framework for classifying reward variables based on the "type" of their reward structure is given. This framework is then used to generate particular variable types that will be considered. In the final section, variables based on a particular type of reward structure which captures information regarding activity completions and numbers of tokens in places are investigated. Example instantiations of variables of these types are given and their relationships to variables used in traditional performance and dependability evaluations are illustrated.

Reward Models

As stated in the introduction, a reward model consists of a stochastic process and a reward structure. The stochastic process represents the dynamics of the system and can be constructed by hand or, automatically, from some network level description. The reward structure is typically a set of one or more functions defined on the states or transitions between states in the process. In all cases known to the author, the interpretation given to each function is either that it is a *rate* at which reward is accumulated or that it is an *impulse* of reward that is obtained at the time of some "event" of the process. These events are typically either entrances to states, exits from states, or transitions between

pairs of states. If the interpretation is of the first type we say that the reward is *rate-based*; reward functions with the second interpretation are said to be *impulse-based.* Performance variables can then be written in terms of the reward structure.

As with the reward structure itself, the manner in which this is done varies greatly in the literature. Variables can be written in terms of the state of the process at a particular time, during an interval of time, or during a time-averaged interval of time. In the first case, the variable typically represents the "status" of the modeled system at some time t and is said to be an *instant-of-time* variable. In the second case, the variable typically represents accumulated benefit derived from operating the system for some interval of time and is said to be an *interval-of-time* variable. If the reward accumulated during some interval is divided by the length of the interval, one obtains a variable which represents the (time-averaged) rate at which reward is accumulated during the interval. Variables of this type are called *time-averaged interval-of-time* variables.

An excellent early exposition of a general reward structure and variable class is given by Howard [38]. In [38], Howard postulates a reward structure on semi-Markov processes that consists of both "yield rates" and "bonuses". In the terminology introduced above, the "yield rates" specify rates at which reward is accumulated and the "bonuses" specify impulses of reward that are obtained at state changes. More precisely, *yield rates* are associated with pairs of states, the interpretation being that for a pair of states i and j, $y_{i,j}(\alpha)$ is rate at which reward is accumulated in state i α time units after i was entered when the successor state is j. Furthermore, *bonuses* are associated with state transitions, where $b_{i,j}(\tau)$ is the reward awarded upon exit from i and subsequent entry into j given that the holding time in i was τ time units. The bonuses paid at state transitions depend both on the transition made and the holding time in the state preceding the transition. The generality of this structure is difficult to fully exploit, due to the complexity of the resulting solution. The analysis required is simplified if one considers reward rates that are constant during the occupancy of each state and bonuses that do not depend on the holding time in the previously occupied state. In this case,

$$y_{i,j}(\alpha) = y_{i,j} \text{ and } b_{i,j}(\tau) = b_{i,j}.$$

Howard then considers the solution for the expected value of an interval-of-time variable

written in terms of reward structures of this type.

Later work concerning reward variables did not make use of reward structure types that were as general as those considered by Howard. In particular, most researchers have limited their attention to a reward structure type with a single function that is rate-based. For example, Meyer [62] provided an early application of a rate-based reward structure to the evaluation of a two-processor system. The variable considered for this application is in the time-averaged interval-of-time category. This work was later extended by Furchtgott and Meyer [25] to provide a solution, albeit computationally expensive, of an interval-of-time variable for systems which are acyclic and nonrecoverable [90] and the reward structure is rate-based. An algorithmic time-domain solution for an interval-of-time variable was then developed by Goyal and Tantawi [31,32] for acyclic rate-based models.

Further work was also limited to rate-based reward structures, but made use of transform techniques to obtain a solution. In particular, Donatiello and Iyer [18,19] provided a closed-form time-domain solution using transform techniques with a rate-based reward structure, thus generalizing the technique described in [62]. Iyer *et al.* [40] later extended this work to obtain a solution in the transform domain for Markov models and an explicit computational formula for each moment of a variable representing accumulated reward over an interval. Kulkarni *et al.* [47] obtained a similar transform solution, but considered models where some transitions cause the loss of all accumulated reward. They also obtained solutions for the distribution of accumulated reward for non-trivial repairable systems by numerical inversion of the transform solution. Most recently, Ciciani and Grassi derived a closed-form solution for general acyclic Markov reward models [13].

While each of these efforts extended known solution techniques for reward models, they did little to extend the generality of reward structure types and hence performance variables that could be considered. Except for the work by Howard, little use has been made of impulse rewards. In addition, the utility of these methods has been limited by having all rewards assigned at the state level. While reasonable for state spaces that are small or have a high degree of regularity, it is often difficult to assign meaningful rewards to large numbers of states. We address these both these issues by 1) constructing general reward structures at the network level and 2) systematically generating variables from

these reward structure types.

The variables that we consider are systematically organized according to reward structure type, category within a reward structure type, and variable type within a category. The manner in which we do this is outlined in Figure 3.1. As depicted in this figure, categories of variables are distinguished at the highest level by the choice of a reward structure type. By type we mean one or more classes of functions that that have a particular interpretation in terms of the networks. For a given reward structure type, variables can be further distinguished by the interval of time that they depend on. Three categories of variables are distinguished at this level, as was discussed earlier in this chapter. The first category, instant-of-time variables, represents the status of the SAN at either a particular time t or in steady state, as shown in Figure 3.1. Interval-of-time and time-averaged interval-of-time variables will also be considered.

Within each of the other two categories, the interval-of-time variables and time-averaged interval-of-time variables, three types of variables are considered. The first type represents the total or time-averaged reward (relative to a particular reward structure) accumulated during some interval $[t, t+l]$. The second type corresponds to an interval of length l as t goes to infinity, and is useful in representing the reward that is accumulated during some interval of finite length in steady-state. The final variable type corresponds to the total or time-averaged reward accumulated during an interval starting at t and of length l as $l \rightarrow \infty$. Thus, as can be seen in Figure 3.1, we consider eight variable types for each reward structure type. We now consider a reward structure type that quantifies benefits associated with activity completions and particular numbers of tokens in places.

Activity-Marking Oriented Variables

A reward structure type which quantifies benefits that can be related to the numbers of tokens in particular places and completions of specific activities is considered in this section. By associating impulse rewards with activity completions, as well as reward rates with particular numbers of tokens in places, we greatly extend the measures of performability that can be considered. Variables based on a reward structure of this type can be used to determine many traditional and non-traditional measures of performance,

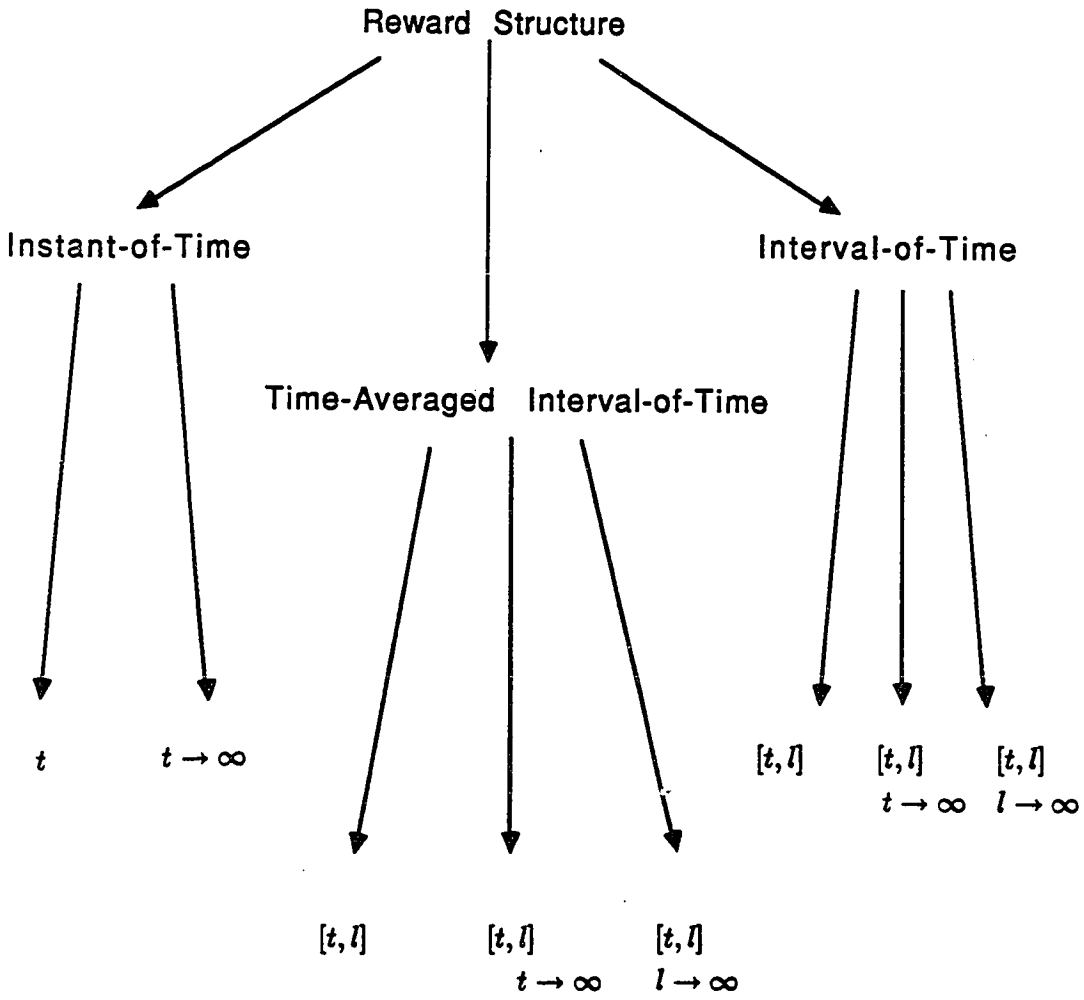


Figure 3.1: Variable Types Considered

including queueing time, queue length, processor utilization, steady-state and interval availability, reliability, and productivity. In addition, if some high-level measure of "worth" is defined, this can be expressed as a particular reward structure of this type.

Structure and Variable Definitions

We define an "activity-marking oriented reward structure" as follows:

Definition III.1 An *activity-marking oriented reward structure* of a stochastic activity network with places P and activities A is a pair of functions:

$\mathcal{C} : A \rightarrow \mathbb{R}$ where for $a \in A$, $\mathcal{C}(a)$ is the reward obtained due to completion of activity a, and

$\mathcal{R} : \mathcal{P}(P, \mathbb{N}) \rightarrow \mathbb{R}$ where for $\nu \in \mathcal{P}(P, \mathbb{N})$, $\mathcal{R}(\nu)$ is the rate of reward obtained when for each $(p, n) \in \nu$, there are n tokens in place p,

where $\mathbb{N}$ is the set of natural numbers and $\mathcal{P}(P, \mathbb{N})$ is the set of all partial functions between P and $\mathbb{N}$.

Informally, impulse rewards are associated with activity completions (via $\mathcal{C}$) and rates of reward are associated with numbers of tokens in sets of places (via $\mathcal{R}$). An element $\nu \in \mathcal{P}(P, \mathbb{N})$ is referred to as a *partial marking.* The marking is partial in the sense that natural numbers are assigned some subset of P, namely the domain of the partial function ν. This assignment is made in a manner identical to the way a (total) marking assigns natural numbers to all the places in the set P. Although $\mathcal{R}$ has a countably infinite domain, the number of elements ν that are of interest to the modeler and, hence, deserving of a non-zero reward assignment will generally be small compared to, say, the number of reachable stable markings of the SAN. Similarly, it will usually be the case that only a fraction of the SAN's activities will have non-zero rewards associated with their completions. We thus use the convention, in practice, that rewards associated with activity completions and partial markings are are taken to be zero if not explicitly assigned otherwise.

Given a SAN with a reward structure of this kind, there are a variety of ways of defining different types of performance (reward) variables, as suggested in the previous section. In particular, we consider two variable types in the instant of time category. The first of these quantifies the behavior of a stochastic activity network at a particular time t. More precisely, if we let V_t denote this variable type then

$$V_t = \sum_{\nu \in \mathcal{P}(P,\mathbb{N})} \mathcal{R}(\nu) \cdot I_t^{\nu} + \sum_{a \in A} C(a) \cdot I_t^{a},$$

where

I_t^{ν} is an indicator random variable representing the event that the SAN is in a marking such that for each $(p, n) \in \nu$, there are n tokens in p at time t, and

I_t^{a} is an indicator random variable representing the event that activity a is the activity that completed most recently at time t.

This variable expresses the total reward (according to the reward structure defined above) associated with a SAN's status at an instant of time t. Depending on the instantiation of the reward structure, the variable can represent a variety of things including queue length and component status (e.g. idle, busy, blocked, failed, functioning). In view of our above observations concerning typical reward structures, and since zero values of $\mathcal{R}(\nu)$ can be ignored in the summation, the number of elements which must be accounted for in this sum is again relatively small.

Depending on the nature of the stochastic activity network in question, I_t^{ν} and I_t^{a} may converge in distribution for all ν and a with non-zero rewards as t approaches ∞. When this happens, the "steady-state" reward obtained at an instant of time can be studied. If we denote the random variable with this steady-state distribution as $V_{t\to\infty}$, its value can be expressed as

$$V_{t\to\infty} = \sum_{\nu \in \mathcal{P}(P,\mathbb{N})} \mathcal{R}(\nu) \cdot I_{t\to\infty}^{\nu} + \sum_{a \in A} C(a) \cdot I_{t\to\infty}^{a},$$

where

$I_{t\to\infty}^{\nu}$ is an indicator random variable representing the event that the SAN is in a marking such that for each $(p, n) \in \nu$, there are n tokens in p in steady-state, and

$I^a_{t\to\infty}$ is an indicator random variable representing the event that activity a is the activity that completed most recently in steady-state.

Variables of the interval and time-averaged-interval categories can also be considered. In these cases, the reward accumulated is related both to the number of times each activity completes and time spent in particular markings during an interval. As was discussed in the previous section, we consider three variable types in each of these categories corresponding to an interval of length l starting at time t ($[t, t+l]$), an interval of length l as $t \to \infty$ ($[t, t+l], t \to \infty$), and an interval starting at t as $l \to \infty$ ($[t, t+l], l \to \infty$). In the following, variable types of the interval category are denoted by "Y" while variables types of the time-averaged category are denoted by "W", each with the appropriate subscript. In particular, let

$$Y_{[t,t+l]} = \sum_{\nu \in \mathcal{P}(P,\mathbb{N})} \mathcal{R}(\nu) \cdot J^{\nu}_{[t,t+l]} + \sum_{a \in A} C(a) \cdot N^a_{[t,t+l]}, \text{ and}$$

$$W_{[t,t+l]} = \frac{Y_{[t,t+l]}}{l}$$

where

$J^{\nu}_{[t,t+l]}$ is a random variable representing the total time that the SAN is in a marking such that for each $(p,n) \in \nu$, there are n tokens in p during $[t, t+l]$, and

$N^a_{[t,t+l]}$ is a random variable representing the number of completions of activity a during $[t, t+l]$.

If $J^{\nu}_{[t,t+l]}$ and $N^a_{[t,t+l]}$ converge in distribution as $t \to \infty$ for all ν and a that have non-zero reward assignments, the time-averaged reward accumulated and total reward accumulated during some interval of length l in steady-state can be studied. If we denote the random variables with these steady-state distribution as $Y_{[t,t+l],t\to\infty}$ and $W_{[t,t+l],t\to\infty}$ then

$$Y_{[t,t+l],t\to\infty} = \sum_{\nu \in \mathcal{P}(P,\mathbb{N})} \mathcal{R}(\nu) \cdot J^{\nu}_{[t,t+l],t\to\infty} + \sum_{a \in A} C(a) \cdot N^a_{[t,t+l],t\to\infty}, \text{ and}$$

$$W_{[t,t+l],t\to\infty} = \frac{Y_{[t,t+l],t\to\infty}}{l},$$

where

$J^{\nu}_{[t,t+l],t\to\infty}$ is a random variable representing the total time that the SAN is in a marking such that for each $(p,n) \in \nu$, there are n tokens in p during a interval of length l in steady-state, and

$N^{a}_{[t,t+l],t\to\infty}$ is a random variable representing the number of completions of activity a during an interval of length l in steady-state.

Similarly, if $J^{\nu}_{[t,t+l]}$ and $N^{a}_{[t,t+l]}$ converge in distribution as $l \to \infty$ for all ν and a that have non-zero reward assignments, the total reward and time-averaged reward accumulated during an infinite interval starting at time t can be expressed as

$$Y_{[t,t+l],l\to\infty} = \sum_{\nu\in\mathcal{P}(P,\mathbb{N})} \mathcal{R}(\nu)\cdot J^{\nu}_{[t,t+l],l\to\infty} + \sum_{a\in A} C(a)\cdot N^{a}_{[t,t+l],l\to\infty}, \text{ and}$$

$$W_{[t,t+l],l\to\infty} = \lim_{l\to\infty} \frac{Y_{[t,t+l]}}{l}$$

where

$J^{\nu}_{[t,t+l],l\to\infty}$ is a random variable representing the total time that the SAN is in a marking such that for each $(p,n) \in \nu$, there are n tokens in p during $[t,\infty)$, and

$N^{a}_{[t,t+l],l\to\infty}$ is a random variable representing the number of completions of activity a during $[t,\infty)$.

Example Variable Instantiations

Traditional measures of dependability and performance can be specified easily using the performance variables just discussed and particular instances of the activity-marking oriented reward structure. To illustrate this, we consider a simple multiprocessor system where all processors service tasks from a single degradable buffer. The normal, fault-free operation of the system is as follows. Tasks arrive as a Poisson process with rate α. If the buffer is full, they are rejected. If not, they are placed in the buffer to be served by the

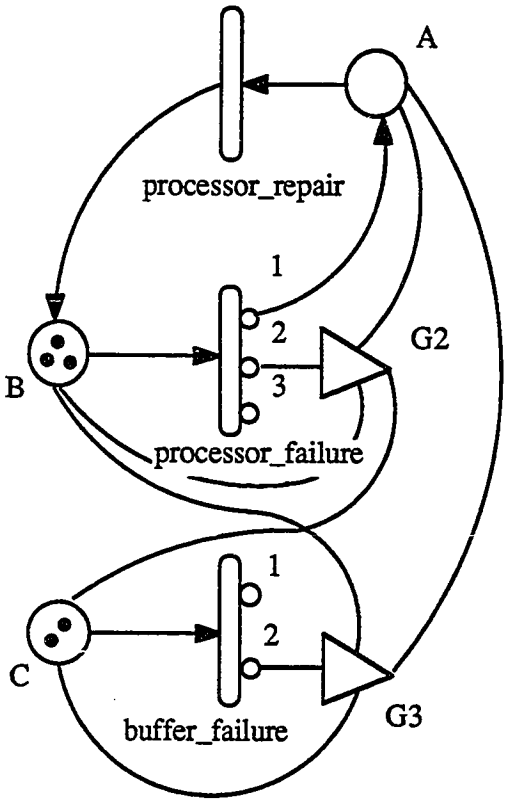
first available processor in a FIFO manner. In addition, processing times are independent and exponentially distributed with each processor having a processing rate β.

Faults can occur both due to a failure of a buffer stage and due to a failure of a processor. In each case, the fault may be covered (i.e. the system degrades successfully to a less productive structure state) or it may result in a total loss of processing capability (i.e. total system failure). Additionally, certain processor failures are repairable. Repairs are performed on one processor at a time, with an exponentially distributed repair time with rate ζ. We assume further that faults in both a buffer stage and a processor occur as Poisson processes with rates λ and γ, respectively.

A stochastic activity network representing changes in the structure of the multiprocessor due to faults is given in Figure 3.2. Since our intent is to illustrate the specification of traditional dependability and performance variables, the model is kept simple. System resources (i.e. processors and buffers) are represented by tokens in places. Place A represents the number of processors queued for repair, place B represents the number of fault-free processors, and place C represents the number of fault-free buffer stages. Activities *processor_failure* and *buffer_failure* represent the occurrence of faults in the processors and buffer stages, respectively. Three types of processor faults are possible, corresponding to the three cases associated with activity *processor_failure*. Case 1 represents the occurrence of a fault that is repairable. Case 2 represents a total system failure, and case 3 represents the occurrence of a non-repairable fault. Cases for *buffer_failure* are similar, except that buffer stages may not be repaired. Here case 1 represents the occurrence of a non-repairable fault and case 2 represents total system failure. Processor repairs are represented by activity *processor_repair*.

Before dependability type measures can be formulated, a definition of "system failure" must be given. In this regard, we say that the system has failed if all processors have failed in a manner such that they cannot be repaired. Then, if we define a reward structure such that

$$C(a) = 0, \quad \forall a \in A$$



Gate	Enabling Predicate	Function
G2	-	MARK(A)=MARK(B)=MARK(C)=0;
G3	-	MARK(A)=MARK(B)=MARK(C)=0;

Activity	Rate	Probability		
		case 1	case 2	case 3
processor_failure	$\gamma * MARK(B)$	cp1	cp2	cp3
buffer_failure	$\lambda * MARK(C)$	cb1	cb2	-
processor_repair	ζ	-	-	-

Figure 3.2: Multiprocessor Fault Model

$$\mathcal{R}(\nu) = \begin{cases} 0 & \text{if } \nu = \{(A,0),(B,0)\} \\ 1 & \text{otherwise,} \end{cases} \tag{3.1}$$

$E[V_t]$ is the reliability (using the above definition of system failure) at time t.

Availability type measures [29,30] can be represented just as easily using this reward structure type and an associated variable. If we consider the system to be available whenever there is at least one processor functioning, the reward structure

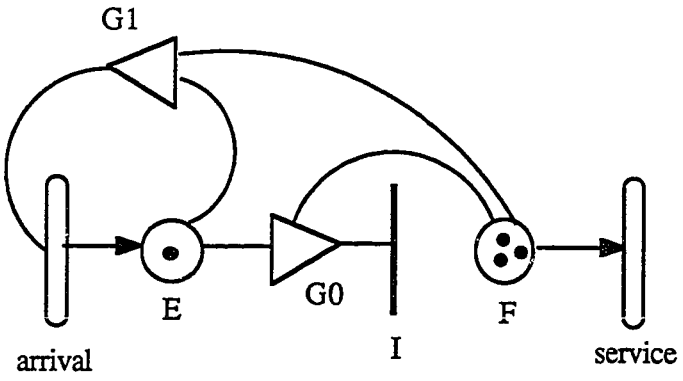
$$C(a) = 0, \quad \forall a \in A$$

$$\mathcal{R}(\nu) = \begin{cases} 0 & \text{if } \nu = \{(B,0)\} \\ 1 & \text{otherwise,} \end{cases} \tag{3.2}$$

can be used to specify availability type measures. For example, using this reward structure, the steady-state availability of the example multiprocessor system is $E[V_{t\to\infty}]$. The interval availability (i.e., the fraction of time the system is available during some interval of length l starting at time t) is $E[W_{[t,l]}]$ using the same reward structure. The distribution of availability, $F(t,l,x)$, is the probability that $W_{[t,l]} \leq x$.

Performance-oriented measures can be specified using a stochastic activity network model that represents task arrivals and completions. A stochastic activity network model of a multiprocessor with N processors and M buffers is given in Figure 3.3. In this figure, each completion of activity *arrival* represents the arrival of a task to the buffer. The buffer is represented by place E. The marking of place E represents the number of tasks queued for service. Place F represents the status of each of the processors, where the number of tokens in F is the number of processors that are busy. The finiteness of the buffer is represented by gate $G1$. Gate $G1$ specifies (via its predicate) that activity *arrival* is enabled only when the number of tasks in the system is less than system capacity (i.e. the sum of the markings of E and F is less than the sum of the number of working processors and buffers).

The service of tasks is represented by activity *service*. Use of a marking dependent activity completion rate for *service* allows us to represent all processors via a single activity,



Gate	Enabling Predicate	Function
G0	MARK(F) < N and MARK(E) > 0	MARK(E) = MARK(E) - 1; MARK(F) = MARK(F) + 1;
G1	MARK(E)+MARK(F) < M+N	identity

Activity	Rate
arrival	α
service	$\beta * MARK(F)$

Figure 3.3: Multiprocessor Performance Model

due to the memoryless property of the exponential distribution. The rate for activity *service* is therefore the number of busy processors multiplied by the rate of a single processor. The number of busy processors is represented by place F.

If we define the throughput of the system during some interval $[t, t+l]$ as the number of tasks that are processed during the interval divided by the length of the interval, the throughput of the example system can be represented using a reward structure consisting only of impulse rewards. Specifically, consider the reward structure

$$C(a) = \begin{cases} 1 & \text{if } a = service \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{R}(\nu) = 0, \quad \forall \nu \in \mathcal{P}(P, \mathbb{N}).$$

Using this reward structure, the throughput is represented by the variable $W_{[t,t+l]}$. Steady-state throughput is given by the limit of this variable when $t = 0$ and $l \to \infty$, i.e. $W_{[0,l]}, l \to \infty$.

An alternate representation of expected steady-state throughput can be formulated based on the arrival rate to the system and the probability that an incoming task is processed. Since tasks arrive as a Poisson process, the probability that an incoming task is processed is one minus the probability the buffer is full. This probability can be captured by the reward structure,

$$C(a) = 0, \quad \forall a \in A$$

$$\mathcal{R}(\nu) = \begin{cases} 1 & \text{if } \nu = \{(E, M)\} \\ 0 & \text{otherwise,} \end{cases}$$

and variable $E[V_{t,t\to\infty}]$. The expected steady-state throughput can then be written as

$$(1 - E[V_{t,t\to\infty}]) \times \alpha,$$

where α is the rate of arrival of tasks to the system.

A representation of expected steady-state response time can be obtained using Little's result [50] and the expected number of tasks in the system in steady-state. The expected

number of tasks in the system can be represented using the reward structure

$$C(a) = 0, \quad \forall a \in A$$

$$\mathcal{R}(\nu) = \begin{cases} i + j & \text{if } \nu = \{(E, i), (F, j)\} \\ 0 & \text{otherwise,} \end{cases}$$

if the variable is taken to be $E[V_{t,t\to\infty}]$. The expected steady-state response time is then $E[V_{t,t\to\infty}]$ divided by the rate at which tasks enter the system, i.e.

$$\frac{E[V_{t,t\to\infty}]}{\alpha}.$$

Processor utilizations can be obtained in a similar manner. Specifically, if the average processor utilization (in steady-state) is defined to be the fraction of the total number of processors that are busy, the reward structure

$$C(a) = 0, \quad \forall a \in A$$

$$\mathcal{R}(\nu) = \begin{cases} i & \text{if } \nu = \{(F, i)\} \\ 0 & \text{otherwise,} \end{cases}$$

can be used. Processor utilization is then $\frac{V_{[t],t\to\infty}}{N}$, where N is the number of processors in the system. As can be seen by the previous examples, traditional performance related and dependability related variables can be easily represented in the reward framework presented. Other performance and dependability related variables can be constructed in a similar manner. In addition, performability measures can be formulated as variables within this type if performance and fault type activities are represented in a single SAN model. An example of a variable such as this will be given in Chapter 5, after a solution method particularly useful for performability models is presented.

CHAPTER IV

CONSTRUCTION TECHNIQUES

Introduction

The network-level representation and performance variables considered in the previous chapters provide a flexible method for representing both systems and questions regarding the performability of those systems. Before the answers to those questions can be found, however, one must consider the development of "construction techniques" for SANs and their associated variables. *Model construction* is the process of identifying a performance variable and determining a base model that permits solution of that variable. As suggested earlier, the construction procedure may depend on the nature of both the variable and stochastic activity network representation employed. SAN characteristics that are important to consider when constructing a base model include both the nature of the activity time distributions and interconnections between model components.

Activity time distributions play an important role in model construction. Specifically, as will be seen in this chapter, the "behavior" of a SAN is Markov only when all its activity time distributions are exponential and its activities are reactivated often enough so that their rates depend only on the current marking. The relative magnitudes of these rates are also important, particularly when representing both dependability and performance related events in a single model. In this case, there may be many orders of magnitude difference between the rates of the fastest and slowest activities. This leads to stochastic process representations that are extremely "stiff" (to be discussed in more detail in the

next section) if traditional construction procedures are employed. One way to deal with this stiffness is through behavioral decomposition. A way to do this at the stochastic activity network level is described in Section 2 of this chapter.

Given that a model is not stiff, several methods can be used to construct a stochastic process representation. In the context of both SANs and other stochastic extensions to Petri nets, this process has typically been obtained by choosing the reachable stable markings of the network to be the states of the process. As alluded to earlier, however, we take a more general view, in which knowledge regarding the structure of the network and performance variables determines the notion of state used. To make this distinction more precise, a stochastic process which supports a large class of variables is referred to as a *detailed base model*, while a stochastic process constructed specifically to support a designated performance variable is a *reduced base model.*

Detailed base models serve two useful functions relative to performability modeling using stochastic activity networks. First, they serve as base models which can support a large class of variables. Second, they aid in proving the correctness of reduced base model construction techniques. Definition of two important detailed base models, proof that they support the variables of Chapter 3, and conditions under which they are Markov are given in Section 3 of this chapter. Procedures to generate the detailed base models are developed when they are Markov. These procedures provide a means to evaluate, analytically, a large class of stochastic activity networks. However, a drawback of the use of detailed base models is that they produce extremely large state spaces when applied to realistic systems.

A construction procedure that produces reduced base models for a restricted, but common, class of SANs and performance variables is considered in Section 4. This class includes stochastic activity networks that have some replicated components, e.g. processors in a multiprocessor or nodes in a computer network, and variables that are regular in the sense that they assign equal rewards to identical events and markings in different replicated components. For these systems, reduced base model construction procedures can be developed that abstract unnecessary information from the base model without rendering it unsolvable. Although the state space savings depends of the degree of replication

present in the system, it typically results in a large reduction in base model size.

Model Decomposition

As stated in the introduction, models that characterize both dependability and performance characteristics of a system typically have great differences in the orders of magnitude in the rates of different activities within the model. This leads (if traditional construction techniques are used) to base model representations that are stiff. Methods to deal with stiff systems include Gear's methods [26], selective randomization [67], and behavioral decomposition [62,65]. The goal of behavioral decomposition is to lump states with high transition rates such that the lumped model is no longer stiff. Performance rates in the lumped states can then be determined via steady-state solution techniques, under the assumption that the consequences of high rate activities approach equilibrium conditions between completions of low-rate activities. Once these rates are determined, performability is evaluated using reward model solution techniques, which will be discussed in the next chapter. All previous work in this area has been done at the state level.

The structure of a stochastic activity network gives us the ability to apply this method earlier in the evaluation procedure, before a stochastic process is generated. When this approach is taken, the activities of the model are decomposed into two sets, depending on their rates of completion. To describe the nature of this decomposition, we introduce the notions of a *performance submodel*, a *structure submodel*, and a set of *common places*. These concepts will enable us to specify necessary and sufficient conditions that a SAN must satisfy if its stochastic behavior is to be described in a manner conforming to "reward model" solution techniques.

Definitions, Conditions, and Procedure

Central to the decomposition is the notion of a set of structure-related activities and a set of performance-related activities. Given a stochastic activity network, the set of *structure-related activities* is the set A_s containing all activities that represent variations

in the system due to a change in system structure. The set of *performance-related activities* is the set A_p containing all activities that represent variations in the internal state and environment of the system, excluding structure-related activities. In terms of A_s and A_p, two submodels are distinguished as follows:

Definition IV.1 Given a stochastic activity network $S = (AN, \mu_0, C, F, G)$, an underlying activity network $AN = (P, A, I, O, \gamma, \tau, \iota, o)$, and set of structure-related activities $A_s \subseteq A$ the *structure submodel* is a stochastic activity network $(M_s, \mu_{0_s}, C_s, F_s, G_s)$ where:

1. $M_s = (P_s, A_s, I_s, O_s, \gamma_s, \tau_s, \iota_s, o_s)$ is an activity network with
   (a) $P_s = \{p \mid p \in P \text{ and } p \in IP(a) \cup OP(a) \text{ for some } a \in A_s\}$,
   (b) A_s is the set of structure-related activities,
   (c) $I_s = \{g \mid g \in I \text{ and } g \in \iota^{-1}(a) \text{ for some } a \in A_s\}$,
   (d) $O_s = \{g \mid g \in O \text{ and } g \in o^{-1}(a, c) \text{ for some } a \in A_s \text{ and } c = 1, 2, \ldots, \gamma(a)\}$, and
   (e) γ_s, τ_s, ι_s, and o_s are the functions γ, τ, ι, and o, respectively, restricted to P_s, A_s, I_s, and O_s.
2. μ_{0_s} is the marking μ_0 restricted to places P_s.
3. C_s is the function C_s restricted to A_s.
4. F_s is the function F restricted to A_s and μ_{0_s}.
5. G_s is the function F restricted to A_s and μ_{0_s}.

Definition IV.2 Given a stochastic activity network $S = (AN, \mu_0, C, F, G)$, an underlying activity network $AN = (P, A, I, O, \gamma, \tau, \iota, o)$, and set of performance-related activities $A_p \subseteq A$ the *performance submodel* is a stochastic activity network $(M_p, \mu_{0_p}, C_p, F_p, G_p)$ where:

1. $M_p = (P_p, A_p, I_p, O_p, \gamma_p, \tau_p, \iota_p, o_p)$ is an activity network with
   (a) $P_p = \{p \mid p \in P \text{ and } p \in IP(a) \cup OP(a) \text{ for some } a \in A_p\}$,
   (b) A_p is the set of performance-related activities,
   (c) $I_p = \{g \mid g \in I \text{ and } g \in \iota^{-1}(a) \text{ for some } a \in A_p\}$,

(d) $O_p = \{g \mid g \in O \text{ and } g \in o^{-1}(a,c) \text{ for some } a \in A_p \text{ and } c = 1,2,\ldots,\gamma(a)\}$, and

(e) γ_p, τ_p, ι_p, and o_p are the functions γ, τ, ι, and o, respectively, restricted to P_p, A_p, I_p, and O_p.

2. μ_{0_p} is the marking μ_0 restricted to places P_p.

3. C_p is the function C restricted to A_p.

4. F_p is the function F restricted to A_p and μ_{0_p}.

5. G_p is the function F restricted to A_p and μ_{0_p}.

Definition IV.3 Given an activity network, the set of *common places* is the set $P_p \cap P_s$.

These two submodels specify stochastic activity networks that may, subject to certain constraints, be solved independently to obtain a system's performability. The constraints restrict the way in which the performance and structure submodel may interact. Specifically,

1. The marking of a common place may not change upon completion of any activity in the performance submodel.

2. No activity $a \in A_s$ may have an activity time distribution function, case distribution, or reactivation function which is dependent on a marking in the set P_p.

3. The completions of activities within the set of performance-related timed activities occur at a sufficiently high rate, as compared to the structure-related timed activities, so that, to a good approximation, the performance submodel will reach steady-state conditions between the occurrence of structure-related activities.

When a SAN meets these conditions, base model construction can proceed in the following manner.

Procedure IV.1 (Constructs a base model of a stochastic activity network using the decomposition technique)

1. Construct a stochastic activity network which meets the previously stated conditions.

2. Select a reward structure type and variable for the structure submodel such that the instantiation of the reward structure type can be determined from the performance submodel.
3. Define a reward structure and (steady-state) performance variable for the performance submodel such that the value of the performance variable serves as input to the reward structure instantiation for the structure submodel.
4. Construct a detailed base model for the structure submodel.
5. For each state of the detailed base model which corresponds to a distinct marking of the common places in the structure submodel:
   (a) Construct a reduced base model for the performance submodel, taking the initial marking of the common places to be their marking in the current state of the detailed base model.
   (b) Solve the reduced base model for the defined performance variable to obtain a value for the structure submodel reward structure.

The state behavior of the structure submodel, together with reward rates determined above, constitute a base model of the system in question. An example of the use of this procedure will be given in Chapter 5 in the section entitled "Solution for Expected Reward", which develops a solution method for reward models of the type constructed by this procedure.

Detailed Base Model Construction

Detailed base models serve two functions relative to performability modeling using stochastic activity networks. Their first function is to serve as a stochastic process representation which can support a large class of variables. Their second function is to aid in proving the correctness of reduced base model construction techniques. To be useful for either of these purposes, they must "support" all the performance variables of interest. By *support* we mean that the variable can be written in terms of state trajectories (sample functions) of the stochastic process which serves as the base model. While this is not a sufficient

condition to insure that the process can be solved for the performance variable, it is a necessary condition. Support of a variable by a stochastic process insures that there is a functional (deterministic) relationship between outcomes of the process (state trajectories) and values of the performance variable.

The "solvability" of a generated base model representation is also an important issue. A base model is *solvable* for a specified variable if the desired probabilistic characteristics of the variable can be obtained from the base model representation. In the context of this dissertation, a base model is solvable (analytically) if it is Markov. In the following subsections, we will consider conditions on a SAN such that its associated detailed base model is Markov, show that these models support the variables defined in Chapter 3, and present procedures for their construction.

Stochastic Process Representations

Two different notions of state will be considered for use in detailed base models. The first, and most detailed, is an "activity-marking state" which captures the dynamics of both markings and activity completions. When information regarding the names of the activities that complete is not needed, a "marking state" can be used to form a detailed base model. Note that even if transitions of the process are considered, a process based on marking states cannot convey information regarding names of activities that complete since different activity completions can result in the same marking state transition. Thus both notions of state are important.

Activity-Marking Behavior

When information regarding both timed activity completions and intervening stable markings is desired, the "activity-marking" behavior of a SAN serves as the detailed base model. This process is based on the notion of an activity-marking state. More precisely, given some SAN, an "am (activity-marking) state" is defined as follows.

Definition IV.4 An *am-state* of a stochastic activity network $SAN = (AN, \mu_0, C, F, G)$, with activities A, is a pair (a, μ) where $a \in A \cup \{\nabla\}$, $\mu \in SR(AN, \mu_0)$ and $\nabla \notin A$. The *initial am-state* is (∇, μ_0).

In terms of the underlying SAN, the first component of an am-state is the most recently completed activity, the second component is the current (stable) marking, and ∇ is a fictitious activity that completes at $t = 0$, bringing the SAN into its initial state. The set of am-states associated with a SAN can be easily constructed from a conceptual point of view. (Looking ahead, however, one would like to avoid actual construction of the complete am-state space since, as is the case with marking-state spaces, they typically convey much more information than needed for the performance variables in question.) Specifically, for a SAN with stable reachable markings $SR(AN, \mu_0)$ and timed activities A let

$$E = \left[\cup_{\mu \in SR(AN,\mu_0)} \cup_{a \in en(\mu)} \cup_{\mu' \in NS(\mu,a)} \{(a, \mu')\}\right] \cup \{(\nabla, \mu_0)\}$$

where $en(\mu)$ is the set of activities enabled in μ and $NS(\mu, a)$ is the set of next stable markings that may be reached upon completion of a in μ. Then E is the set of am-states associated with the SAN. With the preceding notion of state, we are able to define a stochastic process such that a state trajectory of the process captures successive alternations between activities that complete and the stable markings which result. More precisely:

Definition IV.5 The *am-behavior* of a stochastic activity network is a stochastic process $(R, T, L) = \{R_n, T_n, L\}$ where T_n is the time of the *nth* timed activity completion, R_n is the am-state reached after the *nth* timed activity completion, and L is the total number of transitions of the process including the one made at T_0, given that $R_0 = (\nabla, \mu_0)$ and $T_0 = 0$.

A higher level description of behavior can be derived by looking at the am-states as a function of time. Specifically, the *minimal am-behavior of a stochastic activity network* with am-behavior (R, T, L) defined on probability space (Ω, F, P) is the stochastic process $Z = \{Z_t \mid t \in \mathbb{R}^+\}$ where for each $\omega \in \Omega$

$$Z_t(\omega) = \begin{cases} R_{\sup\{n:T_n(\omega)\le t\}}(\omega) & \text{if } t < \sup_{0\le n\le L(\omega)} T_n(\omega) \\ R_{L(\omega)-1}(\omega) & \text{otherwise.} \end{cases}$$

Z is useful when an explicit count of the number of activities that complete is not needed.

The am-behavior is detailed enough to capture both activity and marking related behavior. To see this, consider the notion of an "a-entry" to a stable marking. An *a-entry to* μ is defined to be the event that μ is reached by completion of activity a. Note that the occurrence of an a-entry to μ during execution of the SAN is equivalent to entering the state (a, μ) in the associated am-states. Similarly, the total time spent in μ during $[0, t]$ relative to a-entries to μ is exactly the time spent in (a, μ) during $[0, t]$.

Conditions under which we can solve these base models are now investigated. Informally, the am-behavior of a SAN will be a Markov renewal process [14,15], and the associated minimal am-behavior will be Markov, whenever all timed activities have exponentially distributed activity times and the rates of these distributions can be determined from the current state of the process. Recall that the activity time distribution of an activity (and hence, its rate if it is exponential) is fixed at activation time. Therefore, if the rate is marking dependent, it may depend on a marking other than the current marking, if other activities complete during the time it is active. The following definition of "activation markings" is similar to that defined by Movaghar [70], but is defined at the stochastic activity network level. It allows conditions to be specified to insure that the rates can be determined from the current state of the process.

Definition IV.6 Consider a stochastic activity network with timed activity a which is enabled in a marking μ. An *activation marking of* a *in* μ is a marking μ' such that if the marking of the SAN is μ then a could have previously been activated in μ'.

The set of activation markings of a timed activity in a marking and the nature of the activity time distributions directly affect the nature of the am-behavior. The following theorem identifies stochastic activity networks whose am-behaviors are Markov renewal processes and whose associated minimal am-behaviors are Markov.

Theorem IV.1 Let S be a well specified stochastic activity network with activities which have exponentially distributed activity times. Suppose further that for any timed activity a which is enabled in a marking μ, the rate of completion of a is the same for all activation markings μ' of a in μ and let $r_a(\mu)$ denote this common rate (i.e., for all activation markings μ', $F_a(\mu', x) = 1 - e^{-r_a(\mu)x}$). Then the am-behavior of S is a Markov renewal process with semi-Markov kernel $K = \{K(e_i, e_j, t) \mid e_i, e_j \in E,\ t \in \mathbb{R}^+\}$ where, if $e_i = (a_i, \mu_i)$ and $e_j = (a_j, \mu_j)$,

$$K(e_i, e_j, t) = \frac{r_{a_j}(\mu_i) h_{a_j,\mu_i}(\mu_j)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)} \left(1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i) t}\right).$$

Furthermore, the minimal am-behavior is a Markov process with state-transition rate matrix $[\lambda_{e_i e_j}]$ such that $\lambda_{e_i e_j} = h_{a_j,\mu_i}(\mu_j) r_{a_j}(\mu_i)$, for $i \neq j$, and $\lambda_{e_i e_i} = -\sum_{j \neq i} \lambda_{e_i e_j}$.

Proof:

To show that (R, T, L) is a Markov renewal process we must show that

$$P(R_{n+1} = e_j, T_{n+1} - T_n \leq t \mid R_n = e_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0) =$$

$$P(R_{n+1} = e_j, T_{n+1} - T_n \leq t \mid R_n = e_i) \tag{4.1}$$

for all $n \in \{1, 2, \ldots, L-1\}$, e_i, $e_j \in E$, and $t \in \mathbb{R}^+$. To do this, let

$NA_n(a)$ be the event that the nth activity to complete is a

$NM_n(\mu)$ be the event that the nth marking to be reached is μ

and rewrite the left hand side of (4.1) in terms of this notation. In particular, this expression can be written as

$$P(NA_{n+1}(a_j), NM_{n+1}(\mu_j), T_{n+1} - T_n \leq t \mid R_n = e_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0),$$

which, by the definition of conditional probability, can be re-written as

$$P(NM_{n+1}(\mu_j) \mid NA_{n+1}(a_j), T_{n+1} - T_n \leq t, R_n = e_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0)$$

$$\times\ P(NA_{n+1}(a_j) \mid T_{n+1} - T_n \leq t, R_n = e_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0)$$

$$\times\ P(T_{n+1} - T_n \leq t \mid R_n = e_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0). \tag{4.2}$$

Expression (4.2) is in a form such that each term refers to a particular aspect of SAN execution and, hence, the definition of execution can be used to simplify each term. In particular, note that since the SAN is well specified, the first term is independent of the past history $T_{n+1} - T_n \leq t,\ R_0, \ldots, R_{n-1}, T_0, \ldots, T_n$ and is equal to the probability the marking μ_j is reached given that a_j completes in μ_i (i.e. $h_{\mu_i,a_j}(\mu_j)$). Furthermore, since all activities have exponentially distributed activity times and the rate associated with each activity's distribution is the same in all possible activation markings, the probabilities expressed by the second and third terms of (4.2) are independent of $R_0, \ldots, R_{n-1}, T_0, \ldots, T_n$. In addition, given a current state, the next activity that completes is independent of the time spent in the original state, due to the memoryless property of the exponential distribution. Expression (4.2) can thus be equivalently given as

$$h_{\mu_i,a_j}(\mu_j)P(NA_{n+1}(a_j) \mid R_n = e_i)P(T_{n+1} - T_n \leq t \mid R_n = e_i). \tag{4.3}$$

Probabilistic expressions for the second and third terms of (4.3) can now be derived. In particular, since all activity time distributions are exponential and the rate associated with each activity is the same in all activation markings, the probability that a particular activity a_j completes in a marking μ_i is the fraction of the total rate out of that marking due to a_j, i.e.

$$\frac{r_{a_j}(\mu_i)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)}. \tag{4.4}$$

Furthermore, by the same reasoning, the probability that some activity completes in μ_i during the interval $T_{n+1} - T_n$ (i.e. the third term of (4.3)) is the distribution of the minimum of all the activity times of activities enabled in μ_i. Recall (from [88], for example) that the distribution of $Y_{\min} = \{X_1, X_2, \ldots, X_n\}$ is

$$F_{\min}(t) = 1 - \prod_{i=1}^{n}[1 - F_i(t)].$$

Since the activity time distribution of each activity a is the same in all activation markings of a in μ_i and is given by $1 - e^{-r_a(\mu_i)t}$, it follows that

$$P(T_{n+1} - T_n \leq t \mid R_n = e_i) = 1 - \prod_{a \in en(\mu_i)} e^{-r_a(\mu_i)t}$$

or, equivalently,

$$P(T_{n+1} - T_n \leq t \mid R_n = e_i) = 1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i)t}. \tag{4.5}$$

Substituting (4.4) and (4.5) into (4.3) yields

$$h_{a_j,\mu_i}(\mu_j)\frac{r_{a_j}(\mu_i)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)}\left(1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i)t}\right) \tag{4.6}$$

as a representation for the left hand side of (4.1).

Note that, since all activities have exponentially distributed activity times and the rate associated with each activity's distribution is the same in all activation markings, the derivation of (4.6) did not depend on the conditioning on the past history, and the same result can be obtained for the right hand side. Equation (4.1) thus holds and (R, T, L) is a Markov renewal process. Furthermore, by (4.6), the semi-Markov kernel of the process is

$$K(e_i, e_j, t) = h_{a_j,\mu_i}(\mu_j)\frac{r_{a_j}(\mu_i)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)}\left(1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i)t}\right).$$

Finally, it follows directly from the structure of Z_t (see [14], pg. 316, for example) that the minimal am-behavior is a Markov process with state-transition rate matrix $[\lambda_{e_i e_j}]$ such that $\lambda_{e_i e_j} = h_{a_j,\mu_i}(\mu_j) r_{a_j}(\mu_i)$, for $i \neq j$, and $\lambda_{e_i e_i} = -\sum_{j \neq i} \lambda_{e_i e_j}$. □

Marking Behavior

When information regarding the identities of activities that complete is not needed, a detailed base model with states consisting of the reachable stable markings of the network suffices. In this case, we refer to the states as *m-states*, and the set of m-states (denoted M in the following) is just the set of stable reachable markings of the network. As with am-states, we are interested in two types of behavior. If it is important to retain information regarding the number (but not the names) of activity completions, the "m-behavior" can be studied. Formally,

Definition IV.7 The *m-behavior* of a stochastic activity network is a stochastic process $(R, T, L) = \{R_n, T_n, L\}$ where T_n is the time of the *nth* timed activity completion, R_n is

the m-state reached after the *nth* timed activity completion, and L is the total number of transitions of the process including the one made at T_0, given that $R_0 = (\mu_0)$ and $T_0 = 0$.

Thus the m-behavior is identical to the am-behavior except that the names of activities which complete are not retained. Note however, that the number of activity completions during an interval can by counted. Similarly, activity completions which do not change the marking of the network can be detected since successive times of activity completions are recorded by T_n.

When this level of detail is not needed, the *minimal m-behavior* can serve as a detailed base model. This process is defined in terms of the m-behavior in a manner which is exactly analogous to the way the minimal am-behavior was defined from the am-behavior. In particular, define the *minimal m-behavior of a stochastic activity network* with m-behavior (R, T, L) defined on probability space (Ω, F, P) to be the stochastic process $Z = \{Z_t \mid t \in \mathbb{R}^+\}$ where for each $\omega \in \Omega$

$$Z_t(\omega) = \begin{cases} R_{\sup\{n:T_n(\omega)\leq t\}}(\omega) & \text{if } t < \sup_{0\leq n\leq L(\omega)} T_n(\omega) \\ R_{L(\omega)-1}(\omega) & \text{otherwise.} \end{cases}$$

This is the behavior typically used as a base model by when it is derived from a stochastic Petri net. For example, state behavior defined by Movaghar *et al.* [70] is the minimal m-behavior as we have defined it.

A theorem identifying stochastic networks whose m-behaviors are Markov renewal processes and whose associated minimal m-behaviors are Markov can be written in a manner similar to that for am-states. Specifically,

Theorem IV.2 Let S be a stochastic activity network with activities which have exponentially distributed activity times. Suppose further that for any activity timed a which is enabled in a marking μ, the rate of completion of a is the same for all activation markings μ' of a in μ and let $r_a(\mu)$ denote this common rate (i.e., for all activation markings μ', $F_a(\mu', x) = 1 - e^{-r_a(\mu)x}$). Then the m-behavior of S is a Markov renewal process with

semi-Markov kernel $K = \{K(\mu_i, \mu_j, t) \mid \mu_i, \mu_j \in SR(AN, \mu_0),\ t \in \mathbb{R}^+\}$ where

$$K(\mu_i, \mu_j, t) = \frac{\sum_{a \in en(\mu_i)} r_a(\mu_i) h_{a,\mu_i}(\mu_j)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)} \left(1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i) t}\right).$$

Furthermore, the minimal m-behavior is a Markov process with state-transition rate matrix $[\lambda_{\mu_i \mu_j}]$ such that $\lambda_{\mu_i \mu_j} = \sum_{a \in en(\mu_i)} h_{a,\mu_i}(\mu_j) r_a(\mu_i)$, for $i \neq j$, and $\lambda_{\mu_i \mu_i} = -\sum_{j \neq i} \lambda_{\mu_i \mu_j}$.

Proof:

The proof of this theorem is similar in spirit to that of Theorem IV.1, but differs in that the states of the process are m-states. In particular, to show that (R, T, L) is a Markov renewal process we must show that

$$P(R_{n+1} = \mu_j, T_{n+1} - T_n \leq t \mid R_n = \mu_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0) =$$

$$P(R_{n+1} = \mu_j, T_{n+1} - T_n \leq t \mid R_n = \mu_i) \tag{4.7}$$

for all $n \in \{1, 2, \ldots, L-1\}$, μ_i, $\mu_j \in M$, and $t \in \mathbb{R}^+$. To begin, we note that, by the definition of conditional probability, the left hand side of (4.7) can be re-written as

$$P(R_{n+1} = \mu_j \mid T_{n+1} - T_n \leq t, R_n = \mu_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0)$$

$$\times \quad P(T_{n+1} - T_n \leq t \mid R_n = \mu_i, R_{n-1}, \ldots, R_0, T_n, T_{n-1}, \ldots, T_0). \tag{4.8}$$

As with (4.2), since all activities have exponentially distributed activity times and the rate associated with each activity's distribution is the same in all possible activation markings, the probabilities expressed by the terms of (4.8) are independent of $R_0, \ldots, R_{n-1}, T_0, \ldots, T_n$. Furthermore, given a current state μ_i, the next state that is reached is independent of the time spent in the original state, due to the memoryless property of the exponential distribution. Expression (4.8) can thus be equivalently given as

$$P(R_{n+1} = \mu_j \mid R_n = \mu_i) P(T_{n+1} - T_n \leq t \mid R_n = \mu_i). \tag{4.9}$$

The probabilistic representation of the second term of (4.9) is identical to that given in (4.5). The first term, however, differs in that it represents the probability that μ_j is the next stable marking reached from μ_i, regardless of the activity that completes resulting

in μ_j. Since all activity time distributions are exponential and the rate associated with each activity is the same in all activation markings, this probability is the fraction of the total rate out of μ_i that results in μ_j, i.e.

$$\frac{\sum_{a \in en(\mu_j)} r_a(\mu_i) h_{\mu_i,a}(\mu_j)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)}. \tag{4.10}$$

Substituting (4.10) and (4.5) into (4.9) yields

$$\frac{\sum_{a \in en(\mu_j)} r_a(\mu_i) h_{\mu_i,a}(\mu_j}{\sum_{a \in en(\mu_i)} r_a(\mu_i)} \left(1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i) t}\right) \tag{4.11}$$

as a representation for the left hand side of (4.7).

As in the proof of Theorem IV.1, since all activities have exponentially distributed activity times and the rate associated with each activity's distribution is the same in all activation markings, the derivation of (4.11) does not depend on the conditioning on past history, and the same result can be obtained for the right hand side. Equation (4.7) thus holds and (R, T, L) is a Markov renewal process. Furthermore, by (4.11), the semi-Markov kernel of the process is

$$K(e_i, e_j, t) = \frac{\sum_{a \in en(\mu_j)} r_a(\mu_i) h_{\mu_i,a}(\mu_j)}{\sum_{a \in en(\mu_i)} r_a(\mu_i)} \left(1 - e^{-\sum_{a \in en(\mu_i)} r_a(\mu_i) t}\right).$$

Finally, it follows directly from the structure of Z_t (see [14], pg. 316, for example) that the minimal m-behavior is a Markov process with state-transition rate matrix $[\lambda_{\mu_i \mu_j}]$ such that $\lambda_{\mu_i \mu_j} = \sum_{a \in en(\mu_i)} h_{a,\mu_i}(\mu_j) r_a(\mu_i)$, for $i \neq j$, and $\lambda_{\mu_i \mu_i} = -\sum_{j \neq i} \lambda_{\mu_i \mu_j}$. □

Support of Variables

A stochastic process supports a performance variable if it can be written in terms of state trajectories of the process. Support of the instant-of-time variables is considered first. As discussed in the previous chapter, these variables express the total reward (with respect to a particular instantiation of the reward structure) associated with a SAN's status at some instant of time. The "instant-of-time" nature of these variables allows them to

be supported by the minimal am-behavior. To see this, recall that the notion of state employed here is the am-state, where, for a state (a, μ), a is the most recently completed activity and μ is the stable marking that results from that activity completion. This correspondence allows us to characterize the reward derived from an instant-of-time type variable during an execution of a SAN at the am-state level. Specifically, we define

$$\rho(a, \mu) = \sum_{\nu \in \mathcal{PN}(\mu)} \mathcal{R}(\nu),$$

where $\mathcal{PN}(\mu)$ is the set of $\nu \in \mathcal{P}(P, \mathbb{N})$ such that for each $(p, n) \in \nu$ there are n tokens in place p in marking μ, and

$$\delta(a, \mu) = \mathcal{C}(a).$$

These functions reflect precisely the reward obtained during an execution of a SAN in terms of its minimal am-behavior, and hence, we can write

$$V_t = \rho(Z_t) + \delta(Z_t), \tag{4.12}$$

where $Z = \{Z_t | \ t \in \mathbb{R}^+\}$ is the minimal am-behavior of the SAN in question. Thus V_t is supported for all reward structure choices. If a limiting distribution for Z exists, then $V_{t \to \infty}$ is also supported by the minimal am-behavior, since

$$V_{t \to \infty} = \rho(Z_{t \to \infty}) + \delta(Z_{t \to \infty}),$$

where $Z_{t \to \infty}$ is the steady-state am-state distribution.

Interval-of-time and time-averaged interval-of-time variables are not supported by the minimal am-behavior, since activity completions that do not result in an am-state change cannot be detected by the minimal am-behavior. In this case, the am-behavior is needed. Since this process keeps track of both times of activity completions and resulting stable markings, it can be used to determine the number of completions of each activity during an interval. To show this, define

$$A_t = min\{n \mid T_n > t\}$$

and

$$B_{t+l} = max\{n \mid T_n \leq t + l\}.$$

A_t and B_{t+l} specify the process variables to index over to determine the impulse component of an interval or time-averaged interval variable. Then, if we let ρ and δ be the state-level reward functions defined earlier then the interval-of-time variable, $Y_{t,t+l}$, can be expressed as

$$Y_{t,t+l} = \int_t^{t+l} \rho(Z_t)dt + \sum_{n=A_t}^{B_{t+l}} \delta(R_n)$$

where Z_t is the minimal am-behavior and $\{R_n, T_n, L\}$ is the am-behavior of the SAN of interest. Now, since the minimal am-behavior can be derived from the am-behavior (see Chapter 2), the am-behavior supports all variables of the interval-of-time type. Knowing this, it is apparent that the am-behavior supports time-averaged interval-of-time variables, since

$$W_{[t,t+l]} = \frac{Y_{[t,t+l]}}{l}.$$

It can also be shown, in a manner similar to that of the instant-of-time variables that, if the needed variables converge in distribution as $l \to \infty$ or $t \to \infty$, then the corresponding reward variables are supported. Thus all of the activity-marking oriented variables that were introduced in Chapter 3 are supported by either the am-behavior or minimal am-behavior. Whenever the value of the activity component of the reward structure is identical for all activities, the minimal m-behavior can be used in place of the minimal am-behavior; by the same reasoning the m-behavior can be used in place of the am-behavior. In this case, it is only necessary to keep track of the number of activities that complete during an interval, not the actual identities of the activities that complete.

Construction Procedures

This subsection discusses construction procedures for detailed base models. The goal is to construct a state-level representation that can be solved, via machine implementation, for the desired performance variables. The following construction procedures do this when a network's activity-marking and marking behaviors are Markov renewal processes and the associated minimal behaviors are Markov processes. In this case, the complete behavior (activity-marking or marking) can be characterized by two processes: the minimal

process, and the discrete time imbedded process associated with the complete process. If $K = (s_i, s_j, t)$ is the semi-Markov kernel identified with the complete process, then the discrete time imbedded process associated with the complete process has the state transition probabilities defined by

$$P(s_i, s_j) = \lim_{t \to \infty} K(s_i, s_j, t).$$

It is known (see [14], for example) that the number of visits to a state by the process in the limit is exactly the number of visits to the state by this Markov chain. A characterization of both of these processes can be obtained from the set of reachable states and rates between these states when the behaviors are Markov. In particular, let S be the set of states of the detailed base model employed and, for $s_i, s_j \in S$, λ_{s_i,s_j} be the transition rate between s_i and s_j. Then the transition-rate matrix for the minimal behavior is given by $[\lambda_{i,j}]$, where

$$\lambda_{i,j} = \lambda_{s_i,s_j} \text{ for } i \neq j \quad \text{and} \quad \lambda_{i,i} = -\sum_{j \neq i} \lambda_{s_i,s_j}.$$

Furthermore, the transition matrix of the discrete-time process is given by $[\gamma_{i,j}]$ such that

$$\gamma_{i,j} = \frac{\lambda_{s_i,s_j}}{\sum_{\text{all } s_k \in S} \lambda_{s_i,s_k}}.$$

Thus the minimal process and discrete-time imbedded process may be generated from the set of states and transition rates between these states. Procedures to generate the set of reachable states and rates between these states are now considered.

Marking Behavior

The procedure to generate the marking behavior is the simplest of those considered, and hence described first. The procedure is broken down into two parts, a main and a possible next state generator. The main procedure is given as Procedure IV.2. Here, M denotes the set of reachable m-states, X denotes the set of markings to be expanded, T denotes the set of transition rates between m-states and Y is a set of pairs denoting the set of possible next m-states and rates to these states from a particular marking. Furthermore, the rate from a marking state μ_i to μ_j is denoted as λ_{μ_i,μ_j}.

Procedure IV.2 (Procedure to generate the m-behavior of a stochastic activity network with initial marking μ_0.)

Let $M = \{\mu_0\}$.

Let $X = \{\mu_0\}$.

Let T equal the null set.

While $X \neq$ *null*:

Let $\mu = some\_element\{X\}$.

Let $X = X - \{\mu\}$.

Let $Y = poss\_M(\mu)$.

For each $(y_\mu, y_\lambda) \in Y$:

If $y_\mu \in M$ then

Let $T = T \cup \{(\mu, y_\mu)\}$.

Let $\lambda_{\mu,y_\mu} = y_\lambda$.

else

Let $M = M \cup \{y_\mu\}$.

Let $T = T \cup \{(\mu, y_\mu)\}$.

Let $\lambda_{\mu,y_\mu} = y_\lambda$.

Let $X = X \cup \{y_\mu\}$.

Next $(y_\mu, y_\lambda) \in Y$.

While end.

The procedure is an iterative exploration of the state space that identifies new stable markings that may be reached upon completion of activities. The procedure first initializes the set of m-states and the set of unexpanded states to the initial marking, and the set of transitions to the null set. It then proceeds by exploring each unexpanded marking by generating each possible m-state that can be reached directly from that marking. After

each m-state is generated, one of two things is done. If the m-state has not previously been reached, it is added to the set of reachable m-states and a transition to the state is added to the set of transitions for the process. Finally, since the new state has not yet been expanded, it is added to the list of unexpanded markings. If the marking has been previously reached, a transition is added to the appropriate m-state, but the marking is not added to the list of markings to be expanded. Each marking is removed from the set of unexpanded markings after it is expanded. The procedure terminates when the set of unexpanded markings becomes empty. Note that this will not occur if the set of possible states is not finite.

Two subprocedures are called from the procedure. The first, *some_element*, simply picks an element from a set. The particular element that is picked is not important. The second, *poss_M*, generates the set of m-states (and rates to these states) that are reachable by completion of a single timed activity in a particular marking. Algorithm IV.1 provides this function. In the algorithm, $r_a(\mu)$ denotes the rate of completion of a in μ and $h_{\mu,a}(\mu')$ denotes the probability that μ' is reached upon completion of a in μ.

Algorithm IV.1 (Generates the set, Y, of marking states and rates to these states directly reachable from a particular marking μ.)

Let Y equal the null set.

Let D be the set of activities enabled in μ.

For each $a \in D$:

> Compute the set of possible next stable markings reachable upon completion of a in μ (i.e. $NS(a,\mu)$), the probability of reaching each of these markings (i.e. $h_{\mu,a}$), and check whether this distribution is invariant over possible sets of activity choices. (Use algorithm II.1 or II.3.)
>
> If the distribution is not invariant over possible sets of activity choices then
>
> > Signal SAN is not well specified and abort algorithm.
>
> For each $\mu' \in NS(a,\mu)$:
>
> > If $(\mu', \lambda_{\mu'}) \in Y$ for some $\lambda_{\mu'}$ then
> >
> > > Let $(\mu', \lambda_{\mu'}) = (\mu', \lambda_{\mu'} + r_a(\mu) * h_{\mu,a}(\mu'))$.

else

Let $Y = Y \cup \{(\mu', r_a(\mu) * h_{\mu,a}(\mu'))\}$.

Next $\mu' \in NS(a, \mu)$.

Next $a \in D$.

Activity-Marking Behavior

The procedure to generate the activity-marking behavior is similar to that employed to generate the marking behavior, but can be made more efficient by using information about the behavior of a stochastic activity network. In particular, since the stochastic activity networks we consider have behaviors which are Markov, the next state transition rates depend only on the current marking of the net, and not on the activity that completed resulting in that marking. In terms of an am-state, this means that the next state distribution and rates to these states depend only on the marking component of the originating am-state. Utilization of this fact in the construction procedure results in both time and space savings. First, since the next state behavior from a given state depends only on the marking component of that state, all states with an identical marking component will have identical transition rates out of them. Thus states only need to be expanded if they have a marking component that has not yet been expanded. In addition, this transition information need only be stored once for each set of states having identical marking components. We can thus identify the transition rates for the process with pairs where the first component is a marking and the second component is an am-state.

Procedure IV.3 makes use of these observations. In this procedure, E denotes the set of possible am-states, X denotes the set of markings to be expanded, T denotes the transition rates from markings to am-states and Y is a set of pairs denoting possible next am-states and rates to these states from a particular marking. The rate from a marking μ to an am-state e is written as $\lambda_{\mu,e}$.

Procedure IV.3 (Procedure to generate the am-behavior of a stochastic activity network with initial marking μ_0.)

Let $E = \{(\nabla, \mu_0)\}$.

Let $X = \{\mu_0\}$.

Let T equal the null set.

While $X \neq$ *null*:

Let $\mu = some\_element\{X\}$.

Let $X = X - \{\mu\}$.

Let $Y = poss\_AM(\mu)$.

For each $(y_e, y_\lambda) \in Y$:

If $y_e \in E$ then

Let $T = T \cup \{(\mu, y_e)\}$.

Let $\lambda_{\mu,y_e} = y_\lambda$.

else

Let $E = E \cup \{y_e\}$.

Let $T = T \cup \{(\mu, y_e)\}$.

Let $\lambda_{\mu,y_e} = y_\lambda$.

If no am-state exists in E with marking component identical to y_e

Let $X = X \cup$ marking component of y_e.

Next $(y_e, y_\lambda) \in Y$.

While end.

As with the procedure to generate marking processes, two subprocedures are called. Subprocedure *some_element* is as in the previous procedure. Subprocedure *poss_AM*, on the other hand, differs in that it returns a set of possible next activity-marking states

instead of marking states. Algorithm IV.2 provides this function.

Algorithm IV.2 (Generates the set, Y, of am-states and rates to these states from a particular marking μ)

Let Y equal the null set.

Let D be the set of activities enabled in μ.

For each $a \in D$:

> Compute the set of possible next stable markings reachable upon completion of a in μ (i.e. $NS(a,\mu)$), the probability of reaching each of these markings (i.e. $h_{\mu,a}$), and check whether this distribution is invariant over possible sets of activity choices. (Use algorithm II.1 or II.3.)
>
> If the distribution is not invariant over possible sets of activity choices then
>
> > Signal SAN is not well specified and abort algorithm.
>
> For each $\mu' \in NS(a,\mu)$:
>
> > Let $Y = Y \cup \{((a,\mu'), r_a(\mu) * h_{\mu,a}(\mu'))\}$.
>
> Next $\mu' \in NS(a,\mu)$.

Next $a \in D$.

Reduced Base Model Construction

While detailed models support all of the variables we wish to consider, their use with realistic systems can result in state spaces that are very large. For certain classes of systems, however, reduced base models can be constructed that are much smaller (in number of states) and still preserve system information necessary to solve for the chosen performance variables. This section discusses the theory and application of reduced base model construction methods. These methods are useful for hierarchical systems where, at certain levels in the hierarchy, subsystems are replicated and where contributions to total reward (as defined by an activity-marking variable) are the same for identical subsystems. Such hierarchical systems are becoming increasingly more prevalent and, in a computer

context, include both multiprocessor systems (with many identical processors and busses) and computer networks (with many identical stations).

As with the conditions on system structure, the conditions on the performance variables are not severe. In fact, since the structure of replicated systems is identical, it is natural to assume that possible behaviors of identical subsystems would be treated in an equivalent manner. An example of this would be a computer network consisting of many types of stations, where one is interested in the queue length at a particular type of station, not at an individual station within a type. The next subsection introduces operations that aid in the construction of stochastic activity networks that meet these conditions.

Construction Operations

In order to insure that models built have characteristics that permit construction of a reduced base model, we construct them in a bottom-up manner using two operations. Before we can do this, however, a formal definition of the operands of the construction operations is needed. Specifically, we introduce the notion of a "SAN-based reward model".

Definition IV.8 A SAN S with places P together with a reward structure $(\mathcal{C},\mathcal{R})$ and set of distinguished places $P_D \subseteq P$ is a *SAN-based reward model* $(S,\mathcal{C},\mathcal{R}, P_D)$.

A SAN-based reward model (SBRM) provides us with a structure that can be used to construct larger models that have compact state-space representations. Since the operations discussed below are defined on SAN-based reward models, they operate on the reward structure as well as the network. This insures that the defined variable is supported by the reduced model.

Replicate Operation

The first construction operation replicates a SAN-based reward model. This operation is useful when a system has two or more identical subnetworks. The effect of the operation depends on a set of places which is a subset of the distinguished places of the input SBRM. These places allow communication between the replicated submodels and are not replicated

when the operation is carried out. Informally, the result of the replicate operation on a SAN-based reward model is another SAN-based reward model, where

1. The SAN component is constructed by replicating the original SAN a certain number of times, holding some subset of the distinguished places of the input SBRM common to all replicate SANs, and

2. The reward structure is constructed by assigning an impulse reward to each replicate activity equal to its reward in the original model and reward rates to each partial marking in the new model equal to the rate assigned to the corresponding partial marking in the original model, and

3. The set of distinguished places is the set of places used in the construction operation.

A more formal definition of the replicate operation can be given by considering a particular SAN-based reward model $RM = (S, \mathcal{C}, \mathcal{R}, P_D)$ where S is a SAN with places P and activities A, $(\mathcal{C}, \mathcal{R})$ is an activity-marking oriented reward structure for the SAN, and $P_D \subseteq P$ is a set of distinguished places. Let $\tilde{P}_D \subseteq P_D$ be the set of places to be used in the replicate operation and $RM' = (RM, \tilde{P}_D)$. Now let $\mathcal{RM}$ denote the set of all SAN-based reward models, and $\mathcal{RM}'$ denote the set of all SAN-based reward models together with their possible subsets of distinguished places. Finally, define the function

$$\mathbf{Rep} : \mathcal{RM}' \times I\!N \rightarrow \mathcal{RM}.$$

Rep($(RM, \tilde{P}_D)$, n) is then a SAN-based reward model $(\tilde{S}, \tilde{\mathcal{C}}, \tilde{\mathcal{R}}, \tilde{P}_D)$ with places $\tilde{P}$ and activities $\tilde{A}$ defined as follows:

1. $\tilde{S}$ is a SAN constructed by replicating S n times. All activities and gates are replicated; all places except those in $\tilde{P}_D$ are replicated. Connections to each $p \in \tilde{P}_D$ are identical for each replicated submodel in $\tilde{S}$ and as in S.

2. For each $a \in A$ and corresponding set of replicate activities $a_1, a_2, \ldots, a_n \subseteq \tilde{A}$, $\tilde{\mathcal{C}}(a_i) = \mathcal{C}(a)$ for each replicate submodel i.

3. For each $\nu \in \mathcal{P}(P, I\!N)$ and corresponding partial markings $\nu_i \in \mathcal{P}(\tilde{P}, I\!N)$, $\tilde{\mathcal{R}}(\nu_i) = \mathcal{R}(\nu)$ for each replicate submodel i. $\tilde{\mathcal{R}}(\nu) = 0$ for all other $\nu \in \mathcal{P}(\tilde{P}, I\!N)$.

4. $\tilde{P}_D \subseteq P_D$ is the set of places not replicated.

Join Operation

The replicate operation allows us to construct SAN-based reward models that consist of several identical component SBRMs. This permits the representation of systems that consist of a single replicated structure. Typically, however, distributed systems consist of several different structures, each of which may be replicated. The combination of several different structures is accomplished using the join operation. As with the replicate operation, the join operation both acts on and produces SAN-based reward models. Informally, the effect of the operation is to produce a new SAN-based reward model which is a combination of the individual sub-networks and reward structures. Again, certain places play an important role in the construction operation. In this case, however, a *list* of places is associated with each component SBRM in a manner such that the cardinality of each list of places is the same.

In the joined model, the corresponding places in each list form a single place, with connections to gates and activities as in the individual models. These lists of places thus allow communication between joined subnetworks. As with the replicate operation, each list of places must be a subset of the set of distinguished places of the component SBRM. The reward structure for the new SBRM is constructed in a similar manner to that used for the replication operation. Specifically, the reward structure is constructed by 1) assigning a reward to each activity in the new model equal to the reward of the corresponding activity in the original model and 2) assigning a reward rate for each partial marking in the new model equal to the rate assigned to the corresponding partial marking in the original model.

More formally, let $RM = (S, C, \mathcal{R}, P_D)$ be a SAN-based reward model, where S is a SAN with places P and activities A, $(C, \mathcal{R})$ is a reward structure for the SAN, and P_D is a set of distinguished places. Consider a totally ordered list of l places $P^l = [p_1, p_2, \ldots, p_l]$, where each $p_i \in P_D$, and let $\widehat{RM}^l = (RM, P^l)$. Now let $\mathcal{RM}$ denote the set of all SAN-based reward models, as above, and $\widehat{\mathcal{RM}}^l$ denote the set of all SAN-based reward models

together with possible lists of places of length l. Finally, define the family of functions $\mathbf{Join}_{l,m}$ for possible l and $m \in N$ to be

$$\mathbf{Join}_{l,m} : \overbrace{\widehat{\mathcal{RM}}^l \times \widehat{\mathcal{RM}}^l \times \cdots \times \widehat{\mathcal{RM}}^l}^{m \text{ fold}} \rightarrow \mathcal{RM},$$

where $\mathbf{Join}_{l,m}$ is the join of m SAN-based reward models, each with a list of l distinguished places. Specifically, $\mathbf{Join}_{l,m}((S_1, C_1, \mathcal{R}_1, P_{D1}, P_1^l), (S_2, C_2, \mathcal{R}_2, P_{D2}, P_2^l), \ldots, (S_m, C_m, \mathcal{R}_m, P_{Dm}, P_m^l))$, where $S_1, S_2, \ldots, S_m$ are SANs with places $P_1, P_2, \ldots, P_m$ and activities $A_1, A_2, \ldots, A_m$, is a SAN-based reward model $(\tilde{S}, \tilde{C}, \tilde{\mathcal{R}}, \tilde{P}_D)$ with places $\tilde{P}$ and activities $\tilde{A}$ defined as follows:

1. $\tilde{S}$ is a SAN whose activities and gates are the union of the activities and gates of $S_1, S_2, \ldots, S_m$ respectively, and whose set of places is $P_1 \cup (P_2 - P_2^l) \cup (P_3 - P_3^l) \cup \cdots \cup (P_m - P_m^l)$. Connections to all components are as in the individual models except for connections in submodels $S_2, S_3, \ldots, S_m$ to places $P_2^l, P_3^l, \ldots, P_m^l$ respectively. Connections to these places are made to the respective places in the list P_1^l.

2. For each $a_i \in A_j$, and corresponding $a_i^j \in \tilde{A}$ ($j = 1$ to m), $\tilde{C}(a_i^j) = C(a_i)$.

3. For each $\nu_j \in \mathcal{P}(P_j, I\!N)$, $j = 1$ to m and corresponding $\tilde{\nu} \in \mathcal{P}(\tilde{P}, I\!N)$, $\tilde{\mathcal{R}}(\tilde{\nu}) = \mathcal{R}(\nu_j)$. For each $\tilde{\nu} \in \mathcal{P}(\tilde{P}, I\!N)$ without a corresponding $\nu_j \in \mathcal{P}(P_j, I\!N)$ for some j, $\tilde{\mathcal{R}}(\tilde{\nu}) = 0$.

4. $\tilde{P}_D = P_{D1} \cup (P_{D2} - P_2^l) \cup \ldots \cup (P_{Dm} - P_m^l)$.

These operations can be applied iteratively to build models of systems that are organized in a hierarchical manner. We refer to a SAN-based reward model constructed in this way as a *composed SAN-based reward model.* In order to describe the structure of a composed SAN-based reward model, we make use of a directed tree with three types of nodes: *model/reward structure nodes*, *replicate nodes*, and *join nodes.* See Figure 4.1 for a example of each type of node. Model/reward structure nodes have a degree of zero. These nodes define distinct subnetworks and reward structures in the composed model, and serve as a basis on which to apply the construction operations. Replicate nodes have a degree of one with an outgoing arc that points to the SAN-based reward model that

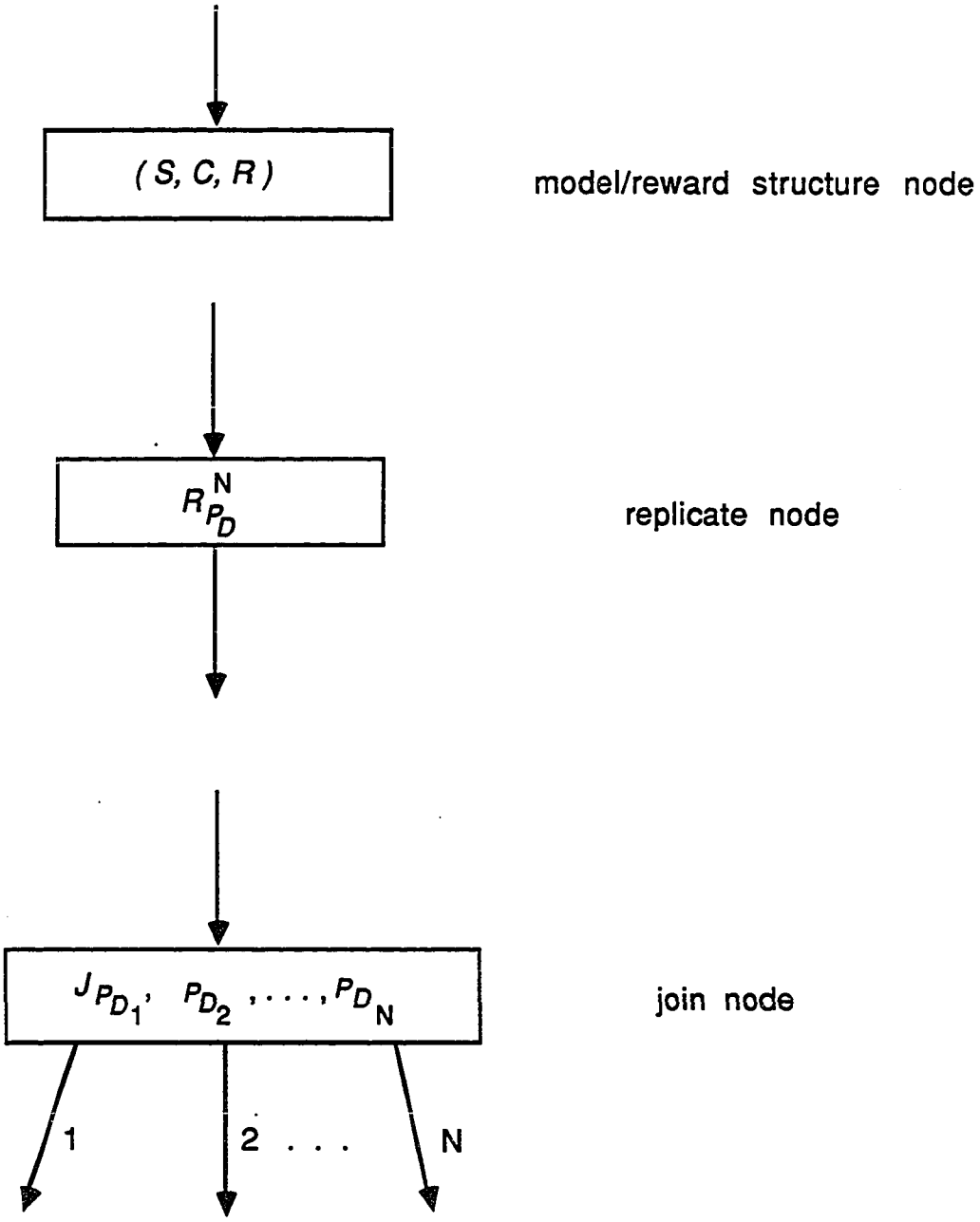


Figure 4.1: Representation of Construction Operations

Model/Reward Structure Node	*Description*
$(S_{BUS}, C_{BUS}, \mathcal{R}_{BUS})$	Multiprocessor Bus Subnetwork
$(S_{PROC}, C_{PROC}, \mathcal{R}_{PROC})$	Processor Subnetwork
$(S_{I/OCHANNEL}, C_{I/OCHANNEL}, \mathcal{R}_{I/OCHANNEL})$	Processor I/O Subnetwork
$(S_{I/OLOGIC}, C_{I/OLOGIC}, \mathcal{R}_{I/OLOGIC})$	I/O Board Logic Subnetwork
$(S_{I/OPORT}, C_{I/OPORT}, \mathcal{R}_{I/OPORT})$	I/O Board Port Subnetwork

Table 4.1: SAN-Based Reward Models for Multiprocessor Example

is replicated. Join nodes have a degree equal to the number of SBRMs that are joined together. In this case, each arc points to a SBRM that is joined.

Information contained inside each node describes the parameters of the particular operation. In particular, each model/reward structure node contains a triple, $(S, C, \mathcal{R})$, specifying the subnetwork and reward structure for the node. The SBRM associated with the node consists of the SAN and reward structure of the node together with a set distinguished places equal to the set of places of the SAN. Replicate nodes contain an integer denoting the number of times the associated SBRM is to be replicated (given as N in the figure) and a set of places to be used in the replicate operation. Join nodes contain an integer (denoting the number of SBRMs to be joined) and a list of places for each SBRM to be joined.

This representation makes it easy to specify multiple applications of replicate and join operations on an initial set of SAN-based reward models. To illustrate this, consider a tightly-coupled multiprocessor system with a single bus which connects five processor cards and two I/O cards together. Furthermore, suppose each processor card has two dedicated I/O channels and each I/O card has eight I/O ports, different in design from the channels on the processor boards. The models of Table 4.1 can then be used to construct a composed SBRM for the multiprocessor using replicate and join structures. Figure 4.2 illustrates the construction of the complete multiprocessor model from these

initial SBRMs.

Representations of State

We now consider possible state-space and, in turn, base model, representations for composed SAN-based reward models. One approach would be to use am or m-states as the state representation, and a detailed base model as the process representation. This, however, leads to excessively large state spaces for many models. The aim, then, is to find one or more alternative representations for state that lead to reduced base models. If support of the variable in question were the only concern, the set of possible values of the variable could be taken as the states of the process. However, the state behavior of this process is not generally Markovian or discrete state, and thus very difficult to solve. On the other hand, if "solvability" were the only issue, the states of the detailed base model could be lumped into a single state in a manner such that the behavior of the reduced base model was easy to deduce. This will not suffice, however, since this process will not generally support a selected variable. Thus both support and solvability are important considerations in selecting a reduced base model representation.

Since we consider a process to be solvable if its behavior is Markov, results regarding lumpings of a Markov process [44] and, equivalently, functions of a Markov process that are Markov [79] are relevant. These results give conditions on a Markov process and a lumping of that process (or equivalently, a functional of that process) such that the new process is also Markov. These conditions are limited, however, in two respects. First, they consider only the solvability of the resulting process; no requirement is made that the resulting process support any particular variable. Second, a direct application would require that the detailed process be generated and that a suitable lumping be found. Machine memory size limitations typical preclude generation of the detailed process and, even if the process could be generated, there is no easy way (without additional information regarding the system being modeled) to determine a suitable lumping from the detailed model.

Stochastic activity networks and the construction operations introduced in the previous subsection provide us with a means to avoid both of these difficulties, due to the way in which the construction operations were defined. In particular, the order of application of

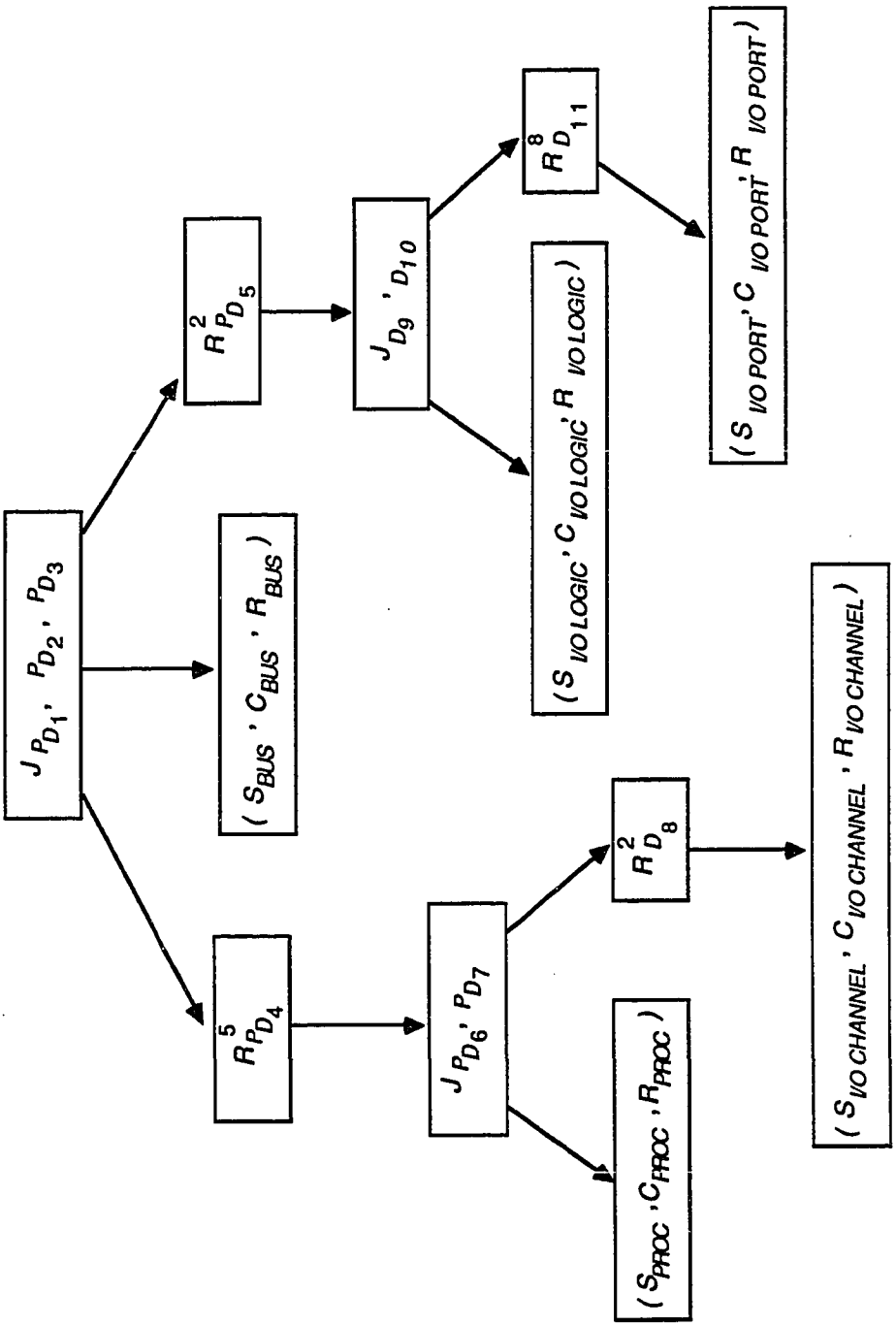


Figure 4.2: Composed SBRM for Multiprocessor Example

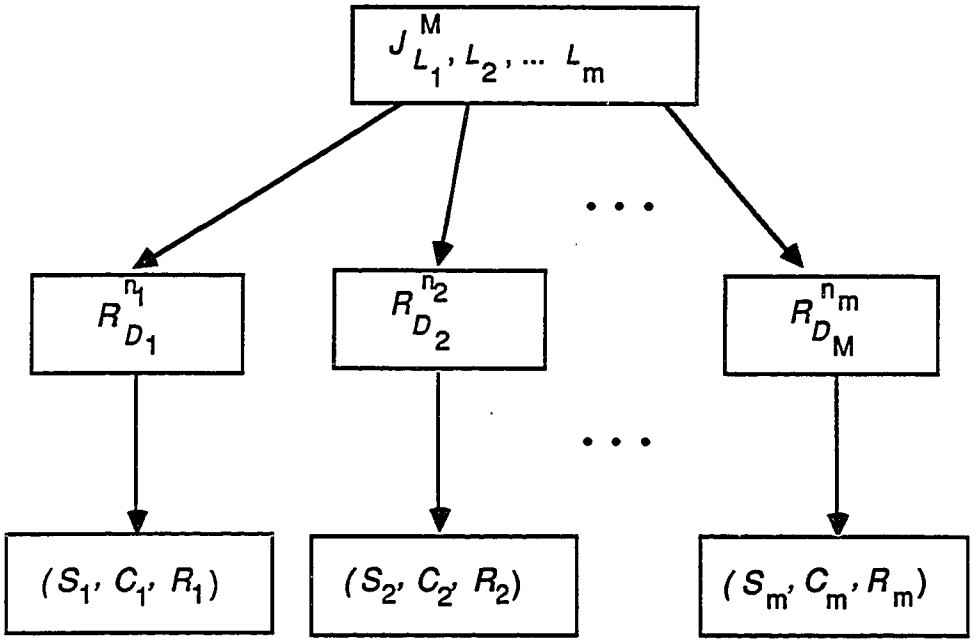


Figure 4.3: Example Composed SAN-Based Reward Model

the operations (as specified by the directed tree representation introduced in the previous subsection) determines a notion of state which both supports the variable in question and is Markov whenever the detailed base model representation is Markov. Informally, this notion of process state is determined such that for each replicate operation, the number of replicate SBRMs in each possible submodel "state" is recorded and for each join operation, a vector of the "state" of each joined submodel is kept. At the lowest level, the "state" of a SBRM model is the marking of the submodel. The complete state for the composed model is then taken to be a pair where the first component is the impulse reward of an activity in the model and the second component is a state formed as described above.

To give a more precise example of how the notion of state would be formulated for a particular application of the replicate and join operations, consider the composed SAN-based reward model of Figure 4.3, where the lowest level nodes are arbitrary models and reward structures. Now let E be the state-space of the am-behavior for this model, and M to be the set of marking components of this am-space. Then define projections to examine the markings of submodel types and individual submodels. Specifically, for each $\mu \in M$

define μ^j to be the projection of (global marking) μ on the places of submodels of type j, $j = 1$ to m. Similarly, define $\mu^{j,k}$ to be the projection of (global marking) μ on the places of submodel k of type j (where places within a submodel have the same ordering).

Furthermore, for each μ^j let $B_{\mu^j} =< \mu^{j,1}, \mu^{j,2}, \ldots, \mu^{j,n_j} >$ be the bag of projected markings for submodels of type j in marking μ. The bag formalism [76] is useful here, since it allows us to compactly characterize the different configurations various replicates of a submodel are in without distinguishing an ordering among them. As will be seen in the following, this representation allows us to specify a functional which, due to the nature of the replicate and join operations, can serve as a reduced base model for the system in question. If, further, we let

$$f(a, \mu) = (\mathcal{C}(a), B_{\mu^1}, B_{\mu^2}, \ldots, B_{\mu^m}) \tag{4.13}$$

then this mapping defines a functional of the minimal and discrete-time embedded process of the am-behavior associated with the SBRM of Figure 4.3. Specifically, if we let Z be the minimal behavior and X be the discrete-time embedded process associated with the am-behavior of the SBRM of Figure 4.3, then

$$U = \{f(Z_t) | \, t \in \mathbb{R}^+\}$$

is the minimal behavior and

$$T = \{f(X_n) \mid n \in \mathbb{N}\}$$

is the discrete-time embedded process of a reduced base model for the SBRM.

Proofs that show that these processes are indeed solvable (Markov) and that they do support the specified performance variable will be given in following sections. Intuitively, however, these models are solvable because replicate submodels of a particular type behave in an equivalent manner when in identical markings. Similarly, since the performance variable was constructed in a bottom up manner as part of the composed SAN-based reward model, it is "symmetric" with respect to different replicates of a given submodel type and thus supported by the reduced base model. Finally, since we can generate these processes without first generating a detailed base model (as shown in the final section of this chapter), we have avoided both of the problems associated with classical lumping methods.

While the functional notion of Equation (4.13) is convenient for two levels of operations, it becomes cumbersome if extended directly to more levels. Instead, we use a set of equations relating SAN-based reward models at one level to those at the next lower level. The nature of this relationship depends on the specific operation used at the particular level. As with the two-level example just given, each replicate node corresponds to a "number of" relation in the state space, and is represented as a bag of submodels at the next lower-level. On the other hand, the "state" of each joined SBRM must be represented explicitly, and hence the operation is represented as a vector of joined submodels. At the lowest level are the submarkings of the individual SAN-based reward models. As with the two-level case discussed above, activities are named in a way which designates the particular submodel that they belong to, and projections of markings on particular submodels are designated by superscripts on the marking (i.e., $\mu^{n_1,n_2,\ldots,n_h}$ denotes the projection of global marking μ on the places of the n_1th SBRM of the highest-level operation, the n_2th SBRM of the next-level operation, and so on).

For example, consider the SAN-based reward model of Figure 4.4. The functional describing the mapping from am-states to reduced states is written as:

$$
\begin{aligned}
f(a,\mu) &= (C(a), V) \\
V &= (B_1, \mu^2) \\
B_1 &= < V_{11}, V_{12} > \\
V_{11} &= (B_{111}, B_{112}) \\
V_{12} &= (B_{121}, B_{122}) \\
B_{111} &= < \mu^{1111}, \mu^{1112} > \\
B_{112} &= < \mu^{1121}, \mu^{1122}, \mu^{1123} > \\
B_{121} &= < \mu^{1211}, \mu^{1212} > \\
B_{122} &= < \mu^{1221}, \mu^{1212}, \mu^{1223} > .
\end{aligned}
$$

Using this notation, the letter B denotes a bag of SBRMs at the next lower level and V denotes a vector. The superscripts on markings denote the marking of a particular sub-

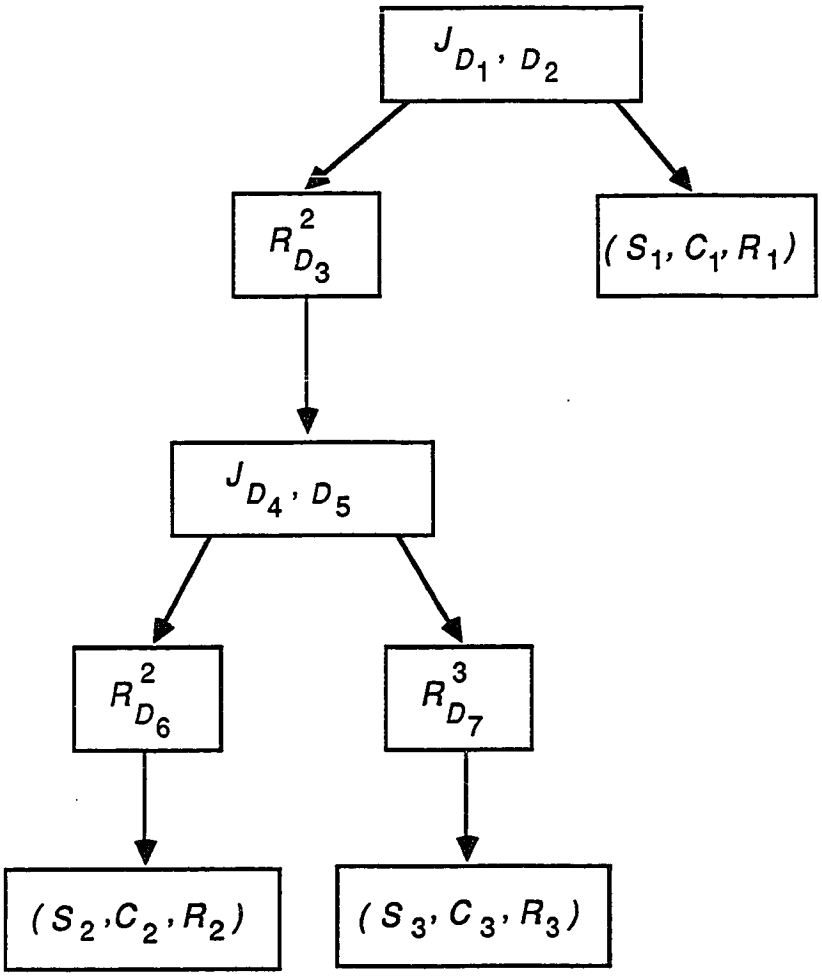


Figure 4.4: Example Composed SAN-based Reward Model

model. This notation is useful in the following sections, where the support and solvability of reduced base models is considered.

Support of Variables

Support of the instant-of-time, interval-of-time, and time-averaged interval-of-time variables will now be considered.

Theorem IV.3 The reduced base model of a composed SBRM supports the instant-of-time, interval-of-time, and time-averaged interval-of-time activity-marking oriented variables defined in Chapter 3.

Proof:

We begin by examining the partition induced on the set of am-states (E) by f. Specifically, define the equivalence relation

$$R_f = \{((b,\mu),(a,\tilde{\mu}))|f(b,\mu) = f(a,\tilde{\mu})\}$$

on E. The equivalence classes of R_f are thus a partition of E. Since the am-behavior supports the instant-of-time, interval-of-time, and time-averaged interval-of-time activity-marking oriented variables (see the section on detailed base models in this chapter), it suffices to show that for all pairs of states e and e' in a common block of the partition, $\delta(e) = \delta(e')$ and $\rho(e) = \rho(e')$. It is immediately apparent from the definition of R_f that $\delta(e) = \delta(e')$; since $f(e) = f(e')$ for all e and e' in a common block, $C(a) = C(a')$, and hence $\delta(e) = \delta(e')$.

To see that $\rho(e) = \rho(e')$, we assume a certain regularity to the application of the replicate and join operations used to build the composed SAN-based reward model. Specifically, we assume, as shown in Figure 4.5, that the directed tree used to represent the construction operation has an even number of levels h, no two replicate or join operations are performed in a row, and the highest node is a join. Furthermore, the number of branches at each node (i.e. the number of submodels replicated or joined together) is as given in the figure. Note that these assumptions result in no loss of generality, since

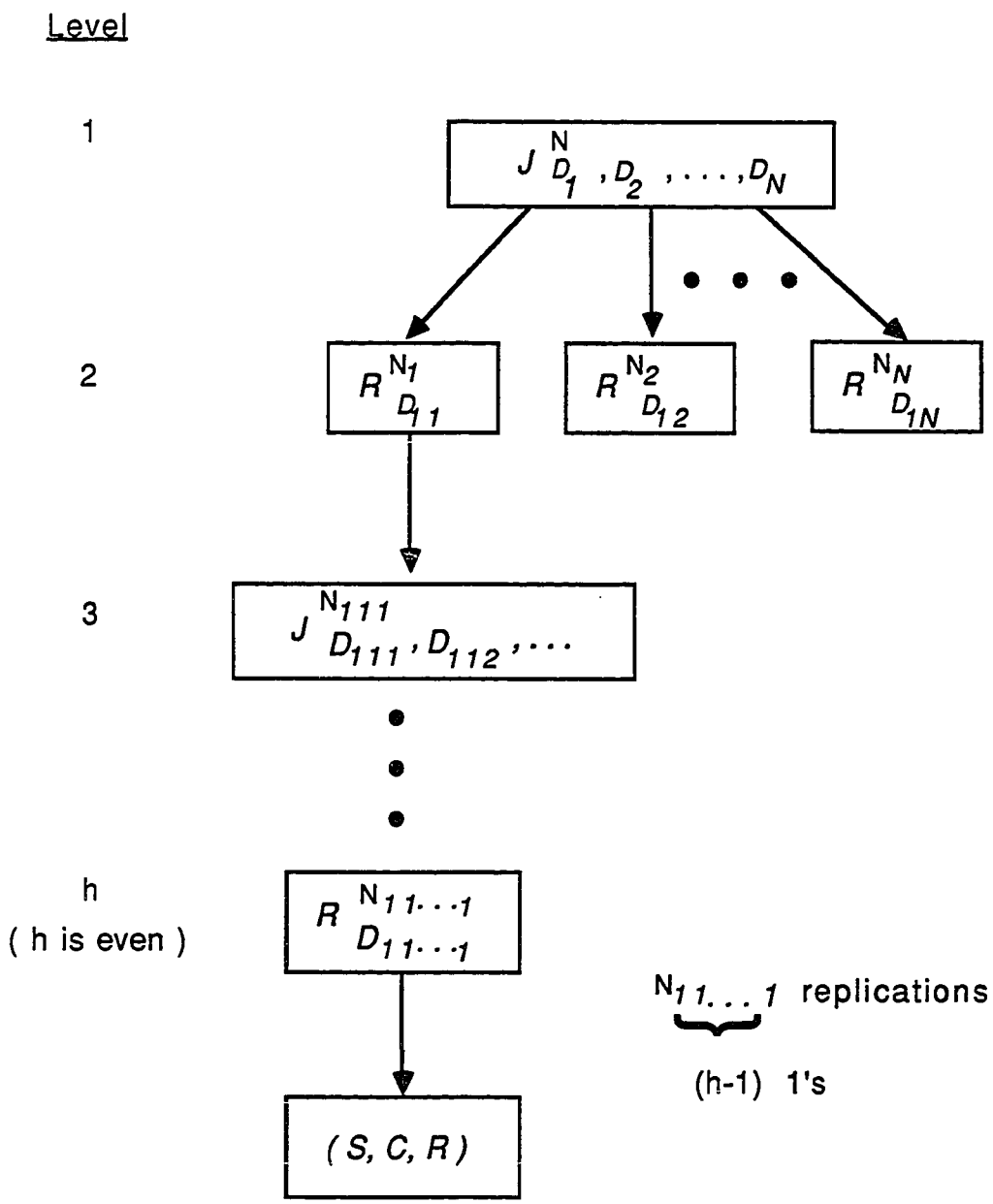


Figure 4.5: "Regular" SAN-Based Reward Model Structure

a join or replication of a single submodel acts as a "no-operation" with respect to the resulting state space and process description. No-operation nodes can thus be added to any arbitrary tree such that it assumes the regular structure of the tree in Figure 4.5.

Now, recall that

$$\rho(b,\mu) = \sum_{\nu \in \mathcal{PN}(\mu)} \mathcal{R}(\nu),$$

which, by the additive nature of $\mathcal{R}(\nu)$ across submodel types and submodels within these types, can be written as

$$\rho(b,\mu) = \sum_{n_1=1}^{N} \sum_{n_2=1}^{N_{n_1}} \sum_{n_3=1}^{N_{n_1,n_2}} \cdots \sum_{n_h=1}^{N_{n_1,n_2,\ldots n_h}} \sum_{\nu \in \mathcal{PN}_{n_1,n_2,\ldots n_h}(b,\mu^{n_1,n_2,\ldots n_h})} \mathcal{R}_{n_1,n_2\ldots n_h}(\nu),$$

where $\mathcal{PN}_{n_1,n_2,\ldots n_h}$ is a function identical to $\mathcal{PN}$ but defined on the submodel $n_1, n_2, \ldots n_h$ and $\mathcal{R}_{n_1,n_2,\ldots n_h}$ is the rate component of the reward structure for the same submodel. Then, since $\mathcal{PN}_{n_1,n_2,\ldots n_h}$ and $\mathcal{R}_{n_1,n_2,\ldots n_h}$ are the same for all replicates of a particular submodel, we can write them as a function that depends only on the particular joins that they belong to. Specifically,

$$\rho(b,\mu) = \sum_{n_1=1}^{N} \sum_{n_2=1}^{N_{n_1}} \sum_{n_3=1}^{N_{n_1,n_2}} \cdots \sum_{n_h=1}^{N_{n_1,n_2,\ldots n_h}} \sum_{\nu \in \mathcal{PN}_{[n_1,n_3\ldots n_{h-1}]}(b,\mu^{n_1,n_2,\ldots n_h})} \mathcal{R}_{[n_1,n_3\ldots n_{h-1}]}(\nu),$$

where $\mathcal{PN}_{[n_1,n_3,\ldots n_{h-1}]}$ and $\mathcal{R}_{[n_1,n_3,\ldots n_{h-1}]}$ are the the $\mathcal{PN}$ and $\mathcal{R}$ functions associated with a particular SBRM type.

Since there may be several replicates of a SBRM in a given "state", we can sum over distinct SBRM "states" (as defined informally earlier) and multiply by the number of SBRMs in that state. This allows us to express ρ as

$$\rho(b,\mu) = \sum_{n_1=1}^{N} \sum_{V_{n_1,n_2} \in B_{n_1}} \#(V_{n_1,n_2}, B_{n_1}) \sum_{n_3=1}^{N_{n_1,n_2}} \cdots \sum_{n_{h-1}=1}^{N_{n_1,n_2,\ldots n_{h-1}}} \sum_{\mu^{n_1,n_2\ldots n_h} \in B_{n_1,n_2,\ldots,n_{h-1}}}$$

$$\#(\mu^{n_1,n_2\ldots n_h}, B_{n_1,n_2,\ldots,n_{h-1}}) \sum_{\nu \in \mathcal{PN}_{[n_1,n_3\ldots n_{h-1}]}(b,\mu^{n_1,n_2,\ldots n_h})} \mathcal{R}_{[n_1,n_3\ldots n_{h-1}]}(\nu),$$

where the B and V are the bags and vectors defined in the previous subsection, and $\#(X, Y)$ denotes the number of occurrences of X in a bag Y.

Furthermore, since, by the definition of f, there are the same number of replicates of each submodel type for all states within a block, for each choice of a particular set of joins, $\rho(e) = \rho(e')$ for all states within a block. Thus the reduced base model supports the activity-marking oriented variables defined in Chapter 3. □

Nature of Stochastic Process

In order for reduced base models to be useful, they must also be solvable. The following theorems establish that the minimal and imbedded discrete-time behaviors of a reduced base model are indeed Markov whenever the am-behavior is Markov. We first establish the following important result.

Theorem IV.4 The minimal behavior (U) of the reduced base model of a composed SBRM is Markov whenever its minimal am-behavior is Markov.

Proof:

To prove the theorem, we employ a continuous-time version of the strong lumpability theorem [89], which states that if the minimal am-behavior is Markov then U is Markov if, for every pair of equivalence classes G, H $(G \neq H)$ of the equivalence relation R_f, the rate from (b, μ) to H is identical for each $(b, \mu) \in G$.

To establish this, we again assume that the SAN-based reward model in question has the regular structure of Figure 4.5. Recall that this can be done without loss of generality. In the context of the proof the assumption is useful, however, since it gives us a specific set of operations to sum over.

We begin by further refining H. H can then be decomposed according to the activity component of the am-states in H. In particular, let

$$H_i^{n_1,n_2,\ldots,n_h} = \{(a, \mu) \mid (a, \mu) \in \bar{H}, \text{ and } a = a_i^{n_1,n_2,\ldots,n_h}\},$$

where $a_i^{n_1,n_2,\ldots,n_h}$ refers to the ith activity in the n_1th SBRM of the first join, the n_2th SBRM of the following replication, and so on. Furthermore, note that in some cases,

$H_i^{n_1,n_2,\ldots,n_h}$ may be empty. The collection of all these non-empty $H_i^{n_1,n_2,\ldots,n_h}$ obviously form a partition of H. Furthermore, to enable us to sum over particular sets of markings, let

$$S_i^{n_1,n_2,\ldots,n_h} = \{\mu' \mid (a,\mu') \in H_i^{n_1,n_2,\ldots,n_h}\}.$$

We now derive an expression for the rate of flow from a state (b,μ) to a state $(a_i^{n_1,n_2,\ldots,n_h},\mu)$ in $H_i^{n_1,n_2,\ldots,n_h}$. If we let $r_{a_i^{n_1,n_2,\ldots,n_h}}(\mu)$ be the rate of completion of activity $a_i^{n_1,n_2,\ldots,n_h}$ in μ and let $h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu')$ be the probability of entering μ' upon completion of $a_i^{n_1,n_2,\ldots,n_h}$ in μ (where $r_{a_i^{n_1,n_2,\ldots,n_h}}(\mu)$ and $h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu')$ are assumed to be zero if $a_i^{n_1,n_2,\ldots,n_h}$ is not enabled in μ), the transition rate from (b,μ) to $(a_i^{n_1,n_2,\ldots,n_h},\mu')$ is just

$$\lambda_{(b,\mu),(a_i^{n_1,n_2,\ldots,n_h},\mu')} = r_{a_i^{n_1,n_2,\ldots,n_h}}(\mu) h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu').$$

But, since the rate of completion of each activity $a_i^{n_1,n_2,\ldots,n_h}$ is the same for all replicates (by definition), the rate depends only on the coordinates denoting instances of the join operations. We can therefore denote the rate for a class of replicate activities as $r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu)$ where $r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu) = r_{a_i^{n_1,n_2,\ldots,n_h}}(\mu)$ for all possible choices of $n_2, n_4, \ldots, n_h$. Furthermore, the rate of an activity depends only on the marking of the submodel in which it is located. We can therefore write $\lambda_{(b,\mu),(a_i^{n_1,n_2,\ldots,n_h},\mu')}$ as

$$\lambda_{(b,\mu),(a_i^{n_1,n_2,\ldots,n_h},\mu')} = r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu'),$$

where $\mu^{n_1,n_2,\ldots,n_h}$ is the projection of global marking μ on the places of n_1th SBRM of the first join, the n_2th replicate of the following replication, and so on. The transition rate to some $H_i^{n_1,n_2,\ldots,n_h}$ is then determined by summing the rate due to all markings that may be reached by completion of activity $a_i^{n_1,n_2,\ldots,n_h}$. This may be expressed as

$\lambda_{(b,\mu),H_i^{n_1,n_2,\ldots,n_h}} =$

$$\sum_{\mu' \in S_i^{n_1,n_2,\ldots,n_h}} r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu').$$

The total rate to a block H from a state (b,μ) can then be obtained by summing over all possible activities in all submodels. To do this we let $A_{[n_1,n_3,\ldots,n_{h-1}]}$ denote the set of

activities for a submodel of type $[n_1, n_3, \ldots, n_{h-1}]$. Then the total transition rate from a state (b, μ) to block H is

$$\lambda_{(b,\mu),H} = \sum_{n_1 \in N} \sum_{n_2 \in N_{n_1}} \cdots \sum_{n_h \in N_{n_1,n_2,\ldots,n_{h-1}}}$$

$$\sum_{i \in A_{[n_1,n_3,\ldots,n_{h-1}]}} r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) \sum_{\mu' \in S_i^{n_1,n_2,\ldots,n_h}} h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu'). \qquad (4.14)$$

We now show that the value given by (4.14) is the same for all states in the equivalence class G. Specifically, consider an arbitrary pair of states $(b, \mu), (c, \tilde{\mu})$ in G. Since (b, μ) and $(c, \tilde{\mu})$ are equivalent (with respect to R_f), the number of submodels in a given state is the same at each level where a replication is done and, hence, there exists a set of permutations on the set of submodels defined by each replicate operation

$$\pi_{n_1} : N_{n_1} \to N_{n_1}, \pi_{n_1,n_2,n_3} : N_{n_1,n_2,n_3} \to N_{n_1,n_2,n_3}, \cdots,$$

$$\pi_{n_1,n_2,\ldots,n_{h-1}} : N_{n_1,n_2,\ldots,n_{h-1}} \to N_{n_1,n_2,\ldots,n_{h-1}},$$

such that

$$\tilde{\mu}^{n_1,n_2,n_3,n_4,\ldots,n_h} = \mu^{n_1,\pi_{n_1}(n_2),n_3,\pi_{n_1,n_2,n_3}(n_4),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}.$$

We write $\pi(\mu)$ to represent the state obtained when these permutations are applied to each submodel (i.e., $\tilde{\mu} = \pi(\mu)$). This fact and Equation (4.14) allow us to write $\lambda_{(c,\tilde{\mu}),H}$ as

$$\lambda_{(c,\tilde{\mu}),H} = \sum_{n_1 \in N} \sum_{n_2 \in N_{n_1}} \cdots \sum_{n_h \in N_{n_1,n_2,\ldots,n_{h-1}}} \sum_{i \in A_{[n_1,n_3,\ldots,n_{h-1}]}}$$

$$r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}) \sum_{\mu' \in S_i^{n_1,n_2,\ldots,n_h}} h_{\pi(\mu),a_i^{n_1,n_2,\ldots,n_h}}(\mu'),$$

which, by rearranging summands, is equal to

$$\lambda_{(c,\tilde{\mu}),H} = \sum_{n_1 \in N} \sum_{\pi_{n_1}^{-1}(n_2) \in N_{n_1}} \cdots \sum_{\pi_{n_1,n_2,\ldots,n_{h-1}}^{-1}(n_h) \in N_{n_1,n_2,\ldots,n_{h-1}}} \sum_{i \in A_{[n_1,n_3,\ldots,n_{h-1}]}}$$

$$r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) \sum_{\mu' \in S_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)} h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)}(\mu').$$

We now show that we can replace the summation over $\mu' \in S_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)$ with a summation over $\pi^{-1}(\mu') \in S_i^{n_1,n_2,\ldots,n_h}$. To do this, we must show that the sets of summands that correspond to non-zero transition (h) functions are equal, i.e.

$$\left\{\mu' \mid \mu' \in S_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h) \text{ and } h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)}(\mu') > 0\right\}$$

$$= \left\{\mu' \mid \pi^{-1}(\mu') \in S_i^{n_1,n_2,\ldots,n_h} \text{ and } h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)}(\mu') > 0\right\}. \quad (4.15)$$

To see this, pick an arbitrary element μ' of the left hand side (LHS) set. Then, since

$$h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)}(\mu') > 0$$

the completion of $a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)$ in $\pi(\mu)$ may result in μ' (i.e., there is a non-zero probability that μ' will be reached upon completion of $a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)$ in $\pi(\mu)$). Since each set of distinguished places at each level where a replication operation is done is a subset of the corresponding set of distinguished places at the next lower level, the effect of the completion of an activity in a submodel is the same, up to a set of permutations on submodels defined by the replicate operations, as the completion of a replicate of that activity in another submodel with the same marking. In other words, if the completion of $a_i^{n_1,n_2,\ldots,n_h}$ in μ may result in μ', then the completion of $a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)$ in $\pi(\mu)$ may result in $\pi(\mu')$. Thus, since the completion of $a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)$ in $\pi(\mu)$ may result in μ', the completion of $a_i^{n_1,n_2,\ldots,n_h}$ in μ may result in $\pi^{-1}(\mu')$. Since μ is an element of G, $\pi^{-1}(\mu') \in S_i^{n_1,n_2,\ldots,n_h}$ and the LHS set is a subset of the right hand side (RHS) set.

Now pick an arbitrary element μ' in the RHS set. Then, note that

$$h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1},n_2,\ldots,n_{h-1}}(n_h)}(\mu') > 0 \text{ iff } h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\pi^{-1}(\mu')) > 0,$$

by the same reasoning used to show the equivalent effect of the completion of replicate activities of replicate submodels in identical markings. Thus, since $\pi^{-1}(\mu') \in S_i^{n_1,n_2,\ldots,n_h}$, the completion of $a_i^{n_1,n_2,\ldots,n_h}$ in μ may result in $\pi^{-1}(\mu')$. Furthermore, by the same argument used in the previous paragraph, the completion of $a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}$ in $\pi(\mu)$ may result in μ'. Since $\pi(\mu) = \tilde{\mu}$ is an element of G, $\mu' \in S_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}$ and the RHS set is a subset of the LHS set. Equation (4.15) therefore holds and we can equivalently express $\lambda_{(c,\tilde{r}),H}$ as

$$\lambda_{(c,\tilde{\mu}),H} = \sum_{n_1 \in N} \sum_{\pi_{n_1}^{-1}(n_2) \in N_{n_1}} \cdots \sum_{\pi_{n_1,n_2,\ldots,n_{h-1}}^{-1}(n_h) \in N_{n_1,n_2,\ldots,n_{h-1}}} \sum_{i \in A_{[n_1,n_3,\ldots,n_{h-1}]}} r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) \sum_{\pi^{-1}(\mu') \in S_i^{n_1,n_2,\ldots,n_h}} h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}}(\mu').$$

By rearranging summands, we obtain

$$\lambda_{(c,\tilde{\mu}),H} = \sum_{n_1 \in N} \sum_{\pi_{n_1}^{-1}(n_2) \in N_{n_1}} \cdots \sum_{\pi_{n_1,n_2,\ldots,n_{h-1}}^{-1}(n_h) \in N_{n_1,n_2,\ldots,n_{h-1}}} \sum_{i \in A_{[n_1,n_3,\ldots,n_{h-1}]}} r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) \sum_{\mu' \in S_i^{n_1,n_2,\ldots,n_h}} h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}}(\pi(\mu')).$$

Since the effect of the completion of an activity in a submodel is the same, up to a set of permutations on submodels defined by the replicate operations, as the completion of a replicate of that activity in a replicate submodel with the same marking,

$$h_{\pi(\mu),a_i^{n_1,\pi_{n_1}(n_2),\ldots,\pi_{n_1,n_2,\ldots,n_{h-1}}(n_h)}}(\pi(\mu')) = h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu').$$

We can thus express $\lambda_{(c,\tilde{\mu}),H}$ as

$$\lambda_{(c,\tilde{\mu}),H} = \sum_{n_1 \in N} \sum_{\pi_{n_1}^{-1}(n_2) \in N_{n_1}} \cdots \sum_{\pi_{n_1,n_2,\ldots,n_{h-1}}^{-1}(n_h) \in N_{n_1,n_2,\ldots,n_{h-1}}} \sum_{i \in A_{[n_1,n_3,\ldots,n_{h-1}]}} r_{a_i^{[n_1,n_3,\ldots,n_{h-1}]}}(\mu^{n_1,n_2,\ldots,n_h}) \sum_{\mu' \in S_i^{n_1,n_2,\ldots,n_h}} h_{\mu,a_i^{n_1,n_2,\ldots,n_h}}(\mu'),$$

establishing the desired result. □

In order to solve for variables that have non-zero rewards associated with activity completions, the discrete-time imbedded behavior of the reduced base model must also be Markov. The following theorem establishes sufficient conditions for this to be the case.

Theorem IV.5 The discrete-time imbedded behavior (T) of the reduced base model of a composed SBRM is Markov whenever its am-behavior is Markov.

Proof:

Proof of this theorem is identical, in approach, to that of Theorem IV.4, and hence will only be sketched. In particular, transition probabilities from states to blocks are computed instead of, but in an identical manner as, transition rates from states to blocks as was done for the minimal behavior. Since these probabilities are the same for every state in a common originating block, a discrete-time version of the strong lumpability theorem [44] can be invoked to show that the discrete-time imbedded behavior of the reduced base model of a composed SAN-based reward model is Markov whenever its am-behavior is Markov. □

Taken together, Theorems IV.4 and IV.5 state that minimal and discrete-time imbedded behaviors of the reduced base model of a composed SBRM are Markov whenever the corresponding am-behavior is Markov. Since these processes also support the variable of interest (Theorem IV.3), they can be used to solve for the variable's probabilistic behavior. Construction procedures which generate a reduced base model are now investigated.

Construction Procedure

While the proceeding theorems state that the reduced base model is indeed solvable whenever the detailed base model is and that the base model supports the specified performance variable, they do not suggest a method for constructing the model. In fact, the proof of solvability relied on lumping arguments. Clearly, one would hope that a procedure could be formulated that did not rely on the prior generation of a detailed base model. This is, in fact, possible, and the resulting procedure is more efficient than those derived for detailed base models.

At a high level, the main procedure is similar to that for the generation of the am-behavior, but operates on reduced states and "representative" markings instead of am-states. A *representative marking of a reduced state* is any marking that maps (via the functional defined earlier) to the marking portion of the reduced state. The representative markings allow the use of the next stable marking generation algorithms developed in Chapter 2, instead of deriving new algorithms that operate directly on reduced states. Once a state has be expanded, the representative marking is no longer needed; hence representative markings need be associated with states only until they are expanded.

In the following procedure, f is the functional defined earlier which maps am-states to reduced states, U is the set of reduced states, X is the set of reduced states yet to be expanded and representative markings for these states, T is the set of transition rates from marking portions of reduced states to reduced states and Y is a set of triples denoting possible next reduced states, representative markings for these states and rates to these states from the marking portion of a reduced state. The marking portion of a reduced state u is denoted as u^m. The rate from a marking portion of a reduced state, u^m, to a reduced state u' is written as $\lambda_{u^m,u'}$.

Procedure IV.4 (Procedure to generate the reduced base model of a stochastic activity network with initial marking μ_0.)

Let $U = \{f(\nabla, \mu_0)\}$.

Let $X = \{(f(\nabla, \mu_0), \mu_0)\}$.

Let T equal the null set.

While $X \neq$ *null*:

Let $(u, \mu) = some\_element\{X\}$.

Let $X = X - \{(u, \mu)\}$.

Let $Y = poss\_U(\mu)$.

For each $(y_u, y_\mu, y_\lambda) \in Y$:

If $y_u \in U$ then

Let $T = T \cup \{(u^m, y_u)\}$.

Let $\lambda_{u^m,y_u} = y_\lambda$.

else

Let $U = U \cup \{y_u\}$.

Let $T = T \cup \{(u^m, y_u)\}$.

Let $\lambda_{u^m,y_u} = y_\lambda$.

If there does not exist a $u \in U$ such that $u^m = y_u^m$

Let $X = X \cup \{(y_u, y_\mu)\}$.

Next $(y_u, y_\mu, y_\lambda) \in Y$.

While end.

Algorithm IV.3 (*poss_U*) generates the set of next possible reduced states and rates to these states from a particular marking portion of a reduced state. This algorithm differs significantly from the corresponding algorithms for detailed base models, since it is not necessary to consider every activity that is enabled in a given marking. Instead, since completions of replicate activities in different submodels with identical submarkings result in the same set of possible next reduced states, it is only necessary to complete one activity within such a set of replicate activities in order to find a set of possible next stable markings. In the following algorithm, the activity that is completed is referred to as the *representative activity.* The number of activities that are in the set of replicate activities is denoted by n. After possible next stable markings and probabilities of these states for each representative activity are computed using Algorithm II.1 or II.3, rates to these states are then found by multiplying the rate due to a single activity by the number of replicate activities in the particular submodel marking. The following algorithm describes this in a more precise manner.

Algorithm IV.3 (Generates the set, Y, of reduced states, representative markings, and rates to these states from a representative marking μ of a particular reduced state.)

Let Y equal the null set.

Let D be the set of pairs (a, n) where a is a representative activity and n is cardinality of the set of replicate activities of a in μ.

For each $(a, n) \in D$:

> Compute the set of possible next stable markings reached upon completion of a in μ (i.e. $NS(a, \mu)$), the probability of reaching each of these markings (i.e. $h_{\mu,a}$), and check whether this distribution is invariant over possible sets of activity choices (Use Algorithm II.1 or II.3)
>
> If the distribution is not invariant over possible sets of activity choices then
>
> > Signal SAN is not well specified and abort algorithm.
>
> For each $\mu' \in NS(a, \mu)$:
>
> > If $\exists (y_u, y_\mu, y_\lambda) \in Y$ such that $f(a, \mu') = y_u$ then
> >
> > $$(y_u, y_\mu, y_\lambda) = (y_u, y_\mu, y_\lambda + r_a(\mu) * n * h_{\mu,a}(\mu'))\}.$$
> >
> > else
> >
> > $$\text{Let } Y = Y \cup \{(f(a, \mu'), \mu', r_a(\mu) * n * h_{\mu,a}(\mu'))\}.$$
>
> Next $\mu' \in NS(a, \mu)$.

Next $(a, n) \in D$.

An example illustrating the effectiveness of reduced base model construction methods will be given in Chapter 6.

CHAPTER V

SOLUTION TECHNIQUES

Introduction

Model solution is the process of determining the probabilistic nature of the selected performance variable from a constructed base model. As alluded to earlier, solution techniques depend on the variable selected, the nature of the associated stochastic activity network, and the construction method employed. At the highest level, solution methods can be differentiated as to whether they use simulation or analysis to determine the nature of the selected variable. Simulation of stochastic activity networks should be done directly at the network level, and a method to do this is described later in this chapter. Simulation can be used to solve for both activity-marking oriented and more general variables.

Analysis techniques differ greatly depending on the constructed representation and selected variable, but typically can be based in traditional stochastic process solution techniques. Broadly speaking, analytical methods for the solution of activity-marking oriented variables depend on the type of the selected variable, although some solution methods may only be applicable to a particular class of reward structure instantiations within a type.

This chapter is organized as follows. The following section briefly outlines methods for the solution for activity-marking oriented variables using known stochastic process solution techniques. No attempt will be made to provide a solution for every reward structure instantiation within every type, but known methods will be applied whenever

possible. The next section details a specific solution method for determining the expected value of interval-of-time and time-averaged-interval-of-time variables of systems that have a finite "lifetime" (with probability one). The *lifetime* of a system is the time after which no further reward is accumulated. "Accumulation" is this context refers to any change in the accumulated reward, including negative as well as positive changes. The final section presents a procedure that can be used to simulate a stochastic activity network at the network level. The level of detail of the simulation is such that all activity-marking oriented variables can be estimated, as well as more general variables. These solution methods, together with implementations of construction methods discussed in the previous chapter, form the basis of an efficient performability evaluation package, which is discussed in the Appendix.

Application of Known Stochastic Process Solution Methods

Once a base model has been constructed for a stochastic activity network, solution techniques depend on the nature of the base model. When its behavior is Markov, many traditional stochastic process solution techniques can be used to determine the probabilistic characteristics of the chosen performance variables. This section will outline some of these techniques and show how they can be applied to the activity-marking oriented variables presented in Chapter 3.

Solution for instant-of-time type variables will be considered first. In this case, the solution techniques depend on whether the variable refers to the status of the SAN at a particular time t or to its status as $t \to \infty$. Solution for the status of a SAN at a particular time t can be accomplished using techniques for transient solution of Markov processes, since instant-of-time variables can be represented as functionals of these processes.

Many methods have been proposed to obtain transient solutions of Markov processes. Perhaps the most direct approach is to recall that (according to Kolmogorov's forward equations [14])

$$\frac{d\pi_t}{dt} = \pi_t A,$$

where A is a state-transition rate matrix and π_t is a row vector representing the probability

of being in each state at time t. Thus (if the state space is finite) the solution for π_t can be obtained using classical differential equation solution methods (e.g. the Runge-Kutta method). These methods, however, are inherently expensive since they do not take advantage of the linear, constant coefficient nature of the state transition-rate matrix [68]. Other methods make use of the observation that

$$\pi_t = \pi_0 e^{At}$$

and attempt to compute e^{At} directly. A good survey of methods to do this is one by Moler and Van Loan [68]. They conclude that 1) the problem is difficult, 2) there is no "best" general purpose method for all classes of matrices, and 3) the utility of a particular method varies depending on the nature of A.

Randomization [33,67] is a particularly suitable method when A is a state transition-rate matrix. The method is based on the subordination of a Markov process to a Poisson process. In particular, if we let P be the transition matrix for the subordinated Markov chain, then $P = \frac{1}{\Lambda}A + I$ where Λ is the rate of the Poisson process. The solution is then obtained by computing the infinite series

$$\pi_t = \sum_{n=0}^{\infty} \pi_0 P^n e^{-\Lambda t} \frac{(\Lambda t)^n}{n!}$$

which, in practice, must be truncated at some point. The error due to truncation varies with the number of terms computed, and can be bounded [33]. It is also possible to investigate the sensitivity of the solution with respect model parameters [35]. Because of these properties, randomization has received increased attention in recent years, and has been incorporated in several performance and/or dependability evaluation packages. For example, it is used both in the System AVailability Estimator (SAVE) [29] to estimate computing system availability and in METASAN (see the Appendix or [81]) to estimate instant-of-time type variables.

Solutions for instant-of-time steady-state variables can make direct use of known stochastic process solution techniques since these variables can be expressed as a function of the long run state-behavior of the base model. The problem, then, is to solve the system of equations

$$\pi A = 0$$

$$\pi(1) + \pi(2) + \ldots + \pi(n) = 1$$

for the row vector π where $\pi(i)$ is the ith element of π. As with the transient solution techniques discussed above, there are many methods to do this; the merits of each depend on the nature of the process being solved [28]. Broadly speaking, these methods can be divided into "direct" and "iterative" methods. *Direct methods* are those that require factorization of the state-transition rate matrix. Examples of direct methods include LU decomposition and QR decomposition [28]. *Iterative methods*, on the other hand, do not require factorization of the state-transition rate matrix, and produce a sequence of approximate solutions, hopefully converging to the exact solution in the limit. The Gauss-Seidel approach is an example of an iterative approach. Both of these general techniques are useful for the solution of steady-state instant-of-time variables.

Direct methods are useful in that they provide exact solutions (allowing for errors due to round off) for all matrices. Their application, however, is limited to systems of moderate size since the solution requires the factorization of the the state transition-rate matrix, which typically requires a large amount of storage. The amount of storage required increases as the factorization operation is performed and the total amount required is hard to predict before performing the factorization. Iterative methods, in contrast, require no more storage space than that required for the original matrix and can, therefore, be used to solve significantly larger systems. They must be used with care, though, since the number of iterations required for convergence depends on the chosen initial condition and, for some systems, they will not converge regardless of the initial condition. Thus neither approach suffices for all matrices, and methods based on both approaches should be included in a evaluation package.

Interval-of-time and time-averaged interval-of-time variables, unlike instant-of-time variables, cannot be written as functionals of the minimal or discrete-time imbedded behavior of a base model. Instead they are "cumulative measures" and are a function of the behavior of the base model over some time period. Solution for these types of variables is, in general, more difficult than solution for variables from the instant-of-time category, and no general solution methods are known that are applicable to all variables within a category. For the most part, however, solution methods developed for more traditional

reward models can be applied to these variables. These methods were surveyed in Chapter 3, from the point of view of the type of reward structure they utilized. From the point of view of solution methods, they fall into several broad classes. At the highest level, the methods are distinguished as to whether they seek to determine the complete probability distribution of a variable or whether they seek to determine only one or more moments of a variable.

Solutions for the expected value and moments of a variable exist for general (cyclic or acyclic) base models which are Markov, although they are generally limited to reward structures that are rate-based. For example, see [40] where the authors provide a solution for the moments (including the expected value) of an interval-of-time variable and a general Markov reward model when the reward structure is rate based. Another example of work in this regard is [52], where the authors provide a closed-form solution for the expected value of interval of time variables when underlying process is acyclic and the reward structure is rate-based.

Three general approaches have been used to obtain the distribution of reward. The first approach in the context of computer systems evaluation was a time-domain approach. Using this approach, the distribution of accumulated reward is formulated as a definite intergal, and the regions of integration are determined from the specific reward rates specified. An early application of this approach was that of Meyer [62], who considered the evaluation of a two-processor, single queue system. Furchtgott and Meyer [24,25] later generalized this approach to provide solutions, using a time-domain approach, for general stochastic processes where the reward rates are non-increasing with time (*non-recoverable*, in the terminology of Wu [89]). Another solution technique that used a similar approach was that by Goyal and Tantawi [31,32], who derived a closed-form solution for general Markov processes when the reward rates were non-increasing with time.

Transform techniques have also been used to determine the distribution of reward. In this case, all efforts have been limited to reward models where the reward structure is rate based. This approach was based on the realization that the distribution of reward could be formulated in a recursive manner, where the reward accumulated during some interval $[0, t]$ given a particular initial state is written in terms of the reward accumulated given

another initial state and a shorter time period. This equation is transformed twice, both on the time and reward variable, to obtain a set of simultaneous equations in the frequency domain. When the process is acyclic, these equations can be inverted to obtain the distribution of reward. Early work using this approach was done by Donatiello and Iyer [18, 20] who used transform techniques to obtain the solution of the n-processor generalization of the queueing system considered by Meyer in [62]. Later work includes that of Iyer *et al.* [40] who obtained a solution in the transform domain for semi-Markov reward models. Most recently, Ciciani and Grassi [13] obtained a solution, using transform techniques, for general acyclic Markov reward models.

All of the efforts described above to obtain the distribution of reward are limited in that they require the underlying base model to be acyclic. No closed-form solution is known for cyclic base models. Although equations in the transform domain can be written for these base models, they cannot be inverted symbolically. In this case, numerical techniques have been used. The approach, in this regard, has been to derive a solution in the transform domain and then invert this solution numerically to obtain the distribution of reward. Algorithms to do this have been developed by Kulkarni *et al.* [47] and Smith [83] and have been applied to the evaluation of multi-processor systems [84]. However, as is noted in [83], these techniques are extremely expensive computationally; the solution of a 365 state system, using the algorithm of [83], requires 1×10^{11} flops and 25 hours on a fast minisupercomputer (a CONVEX C-1).

Solution for Expected Reward

The reward model solution methods surveyed in the previous section are not applicable to variables where reward can be accumulated due to an impulse component as well as rate component. The solution method developed in this section provides a closed-form solution for the expected value of an interval-of-time variable $Y_{0,t}$ where the reward structure has bonuses due to both activity completions (entrances to states) and rates due to numbers of tokens in places (time spent in states).

Derivation

To begin, observe that, if we let ρ and δ be the state-level reward functions defined in Chapter 4, then the reward accumulated during a period of time $[0, t]$ can be expressed as

$$Y_{0,t} = \sum_{u \in U} \rho(u) J_{u,t} + \sum_{u \in U} \delta(u) N_{u,t}$$

where

$J_{u,t}$ is the total time spent in u during $[0, t]$, and

$N_{u,t}$ is the number of entries to u during $[0, t]$, and

U is the state space of the base model (detailed or reduced) of the SAN in question.

Note that, in general, $J_{u,t}$ and $N_{u,t}$ are dependent random variables. However, due to the linearity of the expectation operator,

$$E[Y_{0,t}] = \sum_{u \in U} \rho(u) E[J_{u,t}] + \sum_{u \in U} \delta(u) E[N_{u,t}] \tag{5.1}$$

and, hence, $E[Y_{0,t}]$ can be determined from knowledge of $E[J_{u,t}]$ and $E[N_{u,t}]$.

We now seek practical computational methods to determine $E[Y_{0,t}]$ when the base model process is Markov. By (5.1), this reduces to determining $E[N_{u,t}]$ and $E[J_{u,t}]$ for all u such that $\rho(u)$ or $\delta(u)$ is non-zero.

Solution of $E[J_{u,t}]$ will be considered first. Since knowledge of the number and identity of the activities that complete is not necessary, $E[J_{u,t}]$ can be determined from the minimal behavior of the constructed base model. Conceptually, $E[J_{u,t}]$ could be determined by direct integration by noting that

$$E[J_{u,t}] = E[\int_0^t 1_u(Z_s) ds]$$

where 1_u is an indicator function whose value is 1 if $Z_s = u$ and 0 otherwise. Direct integration is not easy, however, since this requires that the $Pr[Z_t = u]$ be solved as an integrable function of t. Known algebraic solution methods do exist, though, when the

utilization period is unbounded. This algebraic method can be extended to bounded utilization periods by decomposing the problem into two parts, each involving an unbounded utilization. Each part can then be solved separately, and the solutions can be combined to find the solution to the bounded interval problem. Specifically, note that

$$J_{u,\infty} = \int_0^\infty 1_u(Z_s)ds = \int_0^t 1_u(Z_s)ds + \int_t^\infty 1_u(Z_s)ds$$

and, by rearranging terms,

$$J_{u,t} = \int_0^\infty 1_u(Z_s)ds - \int_t^\infty 1_u(Z_s)ds.$$

A further substitution is made by observing that, since Z is time homogeneous (by definition of a SAN)

$$\int_t^\infty 1_u(Z_s)ds = \int_0^\infty 1_u(\bar{Z}^s)ds$$

where $\bar{Z}$ is a process identical to Z except that the initial state distribution of $\bar{Z}$ is the state distribution of Z at time t. This substitution yields:

$$J_{u,t} = \int_0^\infty 1_u(Z_s)ds - \int_0^\infty 1_u(\bar{Z}_s)ds$$

and, taking expectations,

$$E[J_{u,t}] = E[\int_0^\infty 1_u(Z_s)ds] - E[\int_0^\infty 1_u(\bar{Z}_s)ds]. \tag{5.2}$$

Hence $E[J_{e,t}]$ can be determined using methods applicable to unbounded utilization periods.

Methods for determining the times spent in states of a Markov process (in the long run) are well known (see [14], for example) and, hence, will be reviewed here only briefly. Determination of $E[J_{u,\infty}]$ involves determining characteristics of both Z and the embedded discrete time process. The first step is to construct the discrete-time embedded Markov chain (denoted as $\hat{Z}$ in what follows) corresponding to Z. It can then be shown (again, see [14]) for all states which are not absorbing,

$$E[J_{u,\infty}] = \frac{E[\hat{N}_u]}{-\lambda_{u,u}} \tag{5.3}$$

where

$\hat{N}_u$ is the number of times that $\hat{Z}$ enters u during $[0, \infty)$, and

$\lambda_{u,u}$ is the (u, u) entry of the generator matrix of process Z.

The solution of $E[J_{u,\infty}]$ is thus reduced to determining $E[\hat{N}_u]$ for the discrete-time embedded process associated with Z. This is determined by expressing $E[\hat{N}_u]$ in terms of expectations conditioned on the initial state of $\hat{Z}$, i.e.

$$E[\hat{N}_{u_j}] = \sum_{u_i \in U} E[\hat{N}_{u_j} \mid \hat{Z}_0 = u_i] Pr[\hat{Z}_0 = u_i]. \tag{5.4}$$

Methods for determining $E[\hat{N}_{u_j} \mid \hat{Z}_0 = u_i]$ depend on the nature of the process in question. Since we presume a finite lifetime, the processes Z and $\hat{Z}$ have the following properties. If a state is recurrent, the rate component of reward must be zero; otherwise reward could be accumulated indefinitely. By a similar argument, the impulse component of reward for all recurrent states which are not absorbing must also be zero. Absorbing states can have non-zero impulse components, since they are entered at most once. There are no implications regarding transient states; they can have non-zero rate and impulse components. As a consequence of the above, all states for which we need to compute $E[\hat{N}_u]$ are transient, and hence $E[\hat{N}_u]$ is finite. In this case,

$$E[\hat{N}_{u_j} \mid \hat{Z}_0 = u_i] = r_{u_i,u_j} \tag{5.5}$$

where

r_{u_i,u_j} is the (u_i, u_j) entry of the potential matrix of $\hat{Z}$ restricted to non-absorbing states.

The potential matrix of $\hat{Z}$ restricted to non-absorbing states (denoted $\underline{R}$) can be computed directly from its transition matrix. In short (see [14] for details),

$$\underline{R} = (I - \underline{P})^{-1}$$

where

$\underline{P}$ is the transition matrix of $\hat{Z}$ restricted to non-absorbing states, and

I is the identity matrix.

It follows (from (5.3), (5.4) and (5.5)), that

$$E[J_{u,\infty}] = \frac{\pi_0 R(u)}{-\lambda_{u,u}} \tag{5.6}$$

where

π_0 is the row matrix representation of the initial state distribution of Z,

$R(u)$ is the u-th column of the potential matrix of $\hat{Z}$ and is computed by letting $r_{u_i,u_j} = \underline{r}_{u_i,u_j}$ when u_i is not absorbing and $r_{u_i,u_j} = 0$ when u_i is absorbing, and

$\lambda_{u,u}$ is as defined earlier.

Computation of the expected time spent in a state for the case of bounded utilization follows immediately from two applications of (5.6) according to (5.2), and results in the expression

$$E[J_{u,t}] = \frac{(\pi_0 - \pi_t)R(u)}{-\lambda_{u,u}} \tag{5.7}$$

where

π_t is the row matrix representation of the probability distribution of Z_t,

and π_0, $R(u)$, and $\lambda_{u,u}$ are as defined earlier.

It is now clear that, subject to the assumptions previously stated, $E[J_{u,t}]$ can be computed by determination of π_t and $R(u)$ and application of (5.7).

We now derive methods for computation of $E[N_{u,t}]$. $E[N_{u,t}]$ cannot be determined from the minimal behavior of the base model since explicit knowledge of the completions of activities is not preserved in this representation. This information can, however, be inferred from the Markov chain (discrete-time, discrete-state Markov process) associated with the base model behavior. In particular, let

$$P(u_i, u_j) = \lim_{t \to \infty} K(u_i, u_j, t)$$

where

K is the semi-Markov kernel representing the base model behavior of the SAN.

These $P(u_i, u_j)$ are then the transition probabilities for a Markov chain $X = \{X_n \mid n \in N\}$ with state space U. It is known that, in the limit, the number of visits to a state by a Markov renewal process is exactly the number of visits to a state by this Markov chain. $E[N_{u,t}]$ can then be determined by using a technique similar to the one used to determine the expected time spent in a state during a bounded interval. Specifically, observe that for a Markov chain X, the expected number of entries to a state u during $[0,t]$ can be expressed as

$$E[N_{u,t}] = E[\sum_{n=0}^{B_t} 1_u(X_n)]$$

where

B_t is the number of state transitions that occur during $(0,t]$.

As with the explicit formula for the expected time spent in a state, determination of $E[N_{u,t}]$ is not easy since B_t and the X_n are not independent. Note, however, that

$$N_{u,\infty} = \sum_{n=0}^{\infty} 1_u(X_n) = \sum_{n=0}^{B_t} 1_u(X_n) + \sum_{n=B_t+1}^{\infty} 1_u(X_n).$$

Since X is time-homogeneous and the strong Markov property holds at time B_t,

$$\sum_{n=B_t}^{\infty} 1_u(X_n) = \sum_{n=0}^{\infty} 1_u(\bar{X}_n),$$

where $\bar{X}$ is a process identical to X except that the initial state distribution of $\bar{X}$ is the state distribution of X_{B_t}. Application of this substitution and a rearrangement of terms yields

$$N_{u,t} = \sum_{n=0}^{\infty} 1_u(X_n) - \sum_{n=0}^{\infty} 1_u(\bar{X}_n) + 1_u(X_{B_t})$$

and by taking expectations,

$$E[N_{u,t}] = E[\sum_{n=0}^{\infty} 1_u(X_n)] - E[\sum_{n=0}^{\infty} 1_u(\bar{X}_n)] + Pr[X_{B_t} = u]. \tag{5.8}$$

Thus $E[N_{u,t}]$ can be determined from two computations of the number of entries to a state assuming an unbounded utilization period and the probability distribution of X at a single time. As was the case in determining $E[J_{u,t}]$, all states are assumed to be either transient or absorbing. Determination of the number of entries to a state during an unbounded utilization period was considered earlier for transient states (in terms of $\hat{Z}$) and will not be discussed further. When u is absorbing, determination of $E[N_{u,t}]$ is easy and can be determined by observing Z_t. Since each absorbing state can be entered at most once, if u is absorbing,

$$N_{u,t} = \begin{cases} 1 & \text{if } Z_t = u \\ 0 & \text{else} \end{cases}$$

and hence $E[N_{u,t}] = Pr[Z_t = u]$, which can be determined by known methods. The third term of (5.8) can be determined similarly by noting that, since $X_{B_t} = Z_t$, $Pr[X_{B_t} = u]$ $= Pr[Z_t = u]$. These observations permit us to write an explicit formula for $E[N_{u,t}]$ in terms of π_0, π_t, and the potential matrix of X. Specifically,

$$E[N_{u,t}] = \begin{cases} (\pi_0 - \pi_t)R'(u) + \pi_t I(u) & \text{if } u \text{ is transient} \\ \pi_t I(u) & \text{if } u \text{ is absorbing} \end{cases}$$

where

$I(u)$ is a column vector containing a 1 is the u-th position and 0's elsewhere,

$R'(u)$ is the e-th column of the potential matrix of X computed as described for $R(u)$, and

π_0 and π_t are as defined earlier.

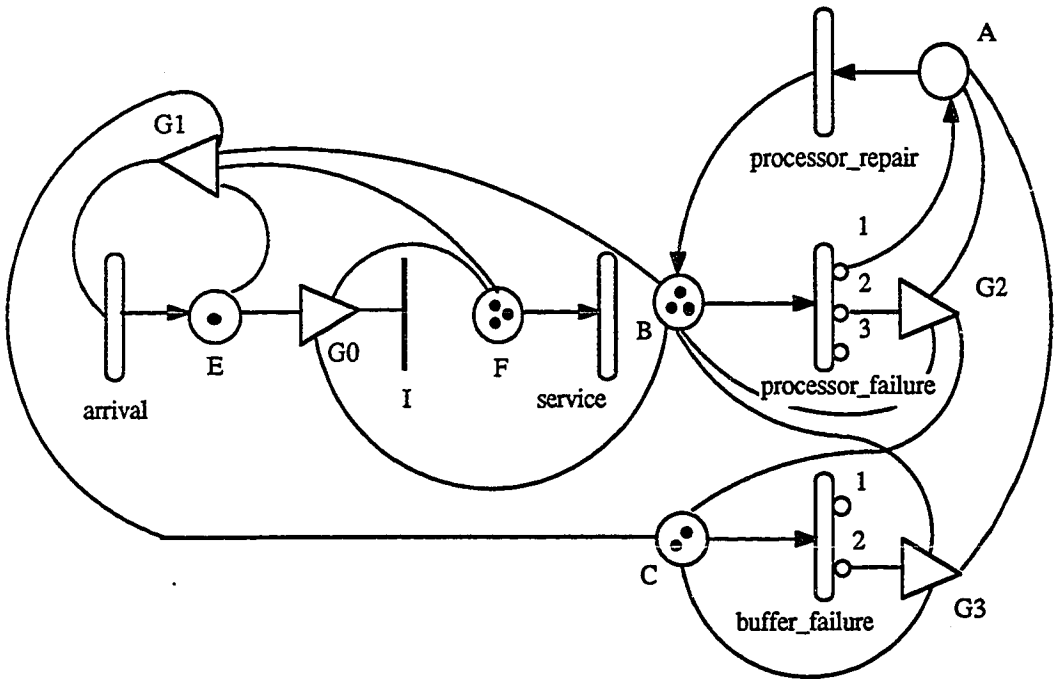
We can now make the appropriate substitutions to obtain an explicit formula for $E[Y_{0,t}]$ in terms of characteristics of the base model, where E_T and E_A denote the sets of transient and absorbing states respectively and all other symbols are as previously defined.

$$E[Y_t] = \sum_{u \in E_T} \left(\frac{\rho(u)(\pi_0 - \pi_t)R(u)}{-\lambda_{u,u}} + \delta(u)((\pi_0 - \pi_t)R'(u) + \pi_t I(u)) \right) + \sum_{u \in E_A} \delta(u)\pi_i I(u)$$

Example

To illustrate the applicability of this method to performability modeling, we consider the performability evaluation of the multiprocessor used in Chapter 3 to illustrate traditional performance and dependability variables. This solution method, together with the decomposition technique developed in Chapter 4, allows us to consider "bottom-line" performability measures that summarize aspects of system performance caused by both fault and workload environments. To see this, we characterize the "total benefit" derived from operating the system for some interval $[t, t+l]$. We assume that "benefit" is derived from the completion of tasks and that costs are associated with the repair of processors. To make the discussion more concrete, we attach a benefit of x dollars to each task completion and a cost of y dollars to each processor repair.

Regarding solution, we construct a performability model which differentiates between "performance" and "structure" related submodels as discussed in Chapter 4. These submodels are just the two SANs considered in the previous examples linked by two common places. A stochastic activity network representing the multiprocessor is given in Figure 5.1. Places B and C are the common places. Since task completions are represented in the performance submodel, the rate of task completions (i.e. throughput) in each structure state serves as the basis for the determination of the rate component of the reward structure. Specifically, the rate of benefit derived for a structure state is the throughput in that state multiplied by the dollar benefit associated with each task completion. Clearly, the throughput is just the arrival rate of tasks to the system multiplied by the probability that a task which arrives will be processed. In terms of the SAN model of the system, an incoming task will be rejected if the sum of the number of tokens in places E and F is equal to the sum of the number of tokens in places B and C (i.e., the system is full). Since tasks arrive as a Poisson process, the probability that an incoming task is processed is one minus the probability that the system is full. This fact allows us to define a reward structure for the performance submodel that permits the determination of system throughput for each structural configuration of the system. In this case, different structural configura-



Gate	Enabling Predicate	Function
G0	MARK(F) < MARK(B) and MARK(E) > 0	MARK(E)=MARK(E)-1; MARK(F)=MARK(F)+1;
G1	MARK(E)+MARK(F) < MARK(B)+MARK(C)	identity
G2	-	MARK(A)=MARK(B)=0; MARK(C)=0;
G3	-	MARK(A)=MARK(B); MARK(C)=0;

Activity	Rate	Probability		
		case 1	case 2	case 3
processor_failure	$\gamma * MARK(B)$	cp1	cp2	cp3
buffer_failure	$\lambda * MARK(C)$	cb1	cb2	-
processor_repair	ζ	-	-	-
arrival	α	-	-	-
service	$\beta * MARK(F)$	-	-	-

Figure 5.1: Degradable Multiprocessor Model

tions are distinguished by the number of functioning buffers and processors. Specifically, when the number of functioning buffers is m, the expected throughput can be obtained using a reward structure where

$$C(a) = 0, \quad \forall a \in A$$

$$\mathcal{R}(\nu) = \begin{cases} 1 & \text{if } \nu = \{(E, m)\} \\ 0 & \text{otherwise,} \end{cases}$$

and taking the variable to be

$$Thru(m, n) = \alpha \times (1 - E_{(m,n)}[V_{t,t\to\infty}]), \tag{5.9}$$

where α is the rate of arrival of tasks to the system and $E_{(m,n)}$ is the expected value of the given variable when in there are m functioning buffers and n functioning processors. Costs associated with processor repairs are represented in the reward structure by associating a reward of $-y$ with each completion of activity *processor_repair*. Under these assumptions, the expected total benefit associated with operating the system for some utilization period $[0, t]$ can be found using the reward structure

$$C(a) = \begin{cases} -y & \text{if } a = processor\_repair \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{R}(\nu) = \begin{cases} x \cdot Thru(m, n) & \text{if } \nu = \{(B, n), (C, m)\} \\ 0 & \text{otherwise,} \end{cases}$$

and variable $E[Y_{[0,t]}]$.

Explicit values for this reward structure can now be obtained by solving the base model associated with the performance submodel to obtain the throughput for each structural configuration. This is done using METASAN[1] (see the Appendix or [81]), a software package that assists in the construction and solution of performability models based on

[1] METASAN is a Trademark of the Industrial Technology Institute

# Buffers	# Processors				
	1	2	3	4	5
0	.833	1.622	2.352	3.008	3.576
1	.968	1.859	2.656	3.338	3.891
2	.994	1.945	2.807	3.532	4.093
3	.999	1.978	2.888	3.658	4.232
4	1.000	1.997	2.934	3.744	4.334
5	1.000	1.999	2.961	3.805	4.412
6	1.000	1.999	2.977	3.850	4.474
7	1.000	1.999	2.986	3.883	4.524
8	1.000	1.999	2.992	3.909	4.566
9	1.000	2.000	2.995	3.928	4.600
10	1.000	2.000	2.997	3.944	4.630

Table 5.1: Throughput as Determined from Performance Submodel

stochastic activity networks. In the case of our example, steady-state state occupancy probabilities for the performance submodel are solved for using the direct steady-state solver in METASAN and throughput is determined using Equation 5.9. Table 5.1 contains the (steady-state) throughput values obtained for different processor-buffer configurations when $\alpha = 5.0$ and $\beta = 1.0$. These throughput values, when multiplied by the benefit associated with the completion of a task, serve as input to the structure submodel reward structure.

For the example in question, the stochastic activity network of the structure submodel and reward structure described earlier provide the input necessary to METASAN to determine the expected number of repairs and expected benefit obtained during some utilization period $[0, t]$. Solution is accomplished using the reward model solution technique just described (implemented as the expected solver in METASAN). Table 5.2 contains the results for various numbers of initial processors and buffers when $t = 240$ and the parameter values of Table 5.3 are used.

Initial #processors	Initial #buffers	$E[Y_{[0,t]}]$
5	0	636.7
5	1	702.9
5	2	746.8
5	3	777.5
5	4	800.0
5	5	817.0
5	6	830.3
5	7	840.9
5	8	849.4
5	9	856.3
5	10	862.1
1	10	189.9
2	10	378.8
3	10	565.7
4	10	739.2
5	10	862.1

Table 5.2: Expected Benefit Derived from Multiprocessor

Parameter Name	Value
γ	.002
λ	.001
ζ	.50
cp1	.80
cp2	.19
cp3	.01
cb1	.99
cb2	.01
x	1.0
y	100.0

Table 5.3: Parameter Values for Degradable Multiprocessor Example

Solution by Simulation

When solution of the desired performance variables can not be obtained by analytic methods, simulation is a viable alternative. Simulation is normally used when activity distributions are not exponential or the state space of the base model is very large or infinite. Conceptually, the simulation can be performed at two levels. The first is the state level. Using this approach, a state level (activity-marking or marking) representation is generated and then simulated, using traditional discrete-event simulation techniques. While this approach has been used for stochastic extensions to Petri nets in the past [21], it is not the most generally applicable method, since it can only be applied to systems that have a finite state spaces. Another approach, and the one we take here, is to perform the simulation directly at the network level. This approach does not require the generation of a state-level representation and permits the study of systems with infinite state spaces.

The method is based on the section in Chapter 2 entitled "Stochastic Activity Network Behavior" which presented an informal description of how a network executes in time by completion of activities (both instantaneous and timed) and the selection of cases. While this description provides insight into the functions of network primitives, it describes the execution of a SAN at a greater level of detail than is necessary to characterize the types of performance variables that we are interested in. Since we are interested in possible sequences of timed activity completions and intervening stable markings, we can characterize the execution of a well specified stochastic activity network in a higher-level manner. At this level, an execution of a SAN can be thought of as a sequence of timed activity-stable marking pairs $(\nabla, \mu_1), (a_2, \mu_2), \ldots$ with the interpretation being that for each activity marking pair (a_i, μ_i), a_i is the timed activity that completed bringing the network into stable marking μ_i. By convention, $\nabla \notin A$ is a fictitious activity that completes at time $t = 0$ bringing the network into its initial marking.

Since the SANs we are interested in are well specified, trajectories of the type just described can be computed directly, without considering possible completions of intervening instantaneous activities, once the next stable marking probability distributions are known. These distributions could be computed *a priori*, using Algorithm II.2, if the set of reachable markings for the SAN is finite. However, since we want to consider SANs

whose sets of reachable markings are not finite, we compute $h_{\mu,a}$ using Algorithm II.1 or II.3 the first time marking μ is reached and timed activity a completes. This result can then be saved and used whenever μ is reached and a completes in the future.

A procedure that generates trajectories is now presented, which is suitable to implementation by simulation. The term *potential completion time* refers to a time selected, stochastically, from an activity time distribution function, and is the time the activity will complete, if is it not interrupted.

Procedure V.1 (Generates an execution of a stochastic activity network)

Set *list_of_active_activities* to *null.*

Set *cur_marking* to *initial_marking.*

Generate *potential_completion_time* for each activity which may complete in the *cur_marking* and add to *list_of_active_activities.*

While *list_of_active_activities* $\neq$ *null*:

> Set *cur_activity* to activity with earliest *potential_completion_time.*
>
> Remove *cur_activity* from *list_of_active_activities.*
>
> If $h_{cur\_marking,cur\_activity}$ has not yet been computed then
>
> > Compute $h_{cur\_marking,cur\_activity}$.
>
> Select new *cur_marking* probabilistically from $NS(cur\_marking, cur\_activity)$ using $h_{cur\_marking,cur\_activity}$.
>
> Remove all activities from *list_of_active_activities* that are not enabled in *cur_marking.*
>
> Remove all activities from the *list_of_active_activities* for which the *cur_marking* is a reactivation marking.
>
> Select a *potential_completion_time* for all activities which are enabled but not on the *list_of_active_activities* and add them to the list.

While end.

A procedure such as this can be used to estimate the values of many performance variables, including the variables in activity-marking category, as well as other more general variables. Specific performance variables, of the activity-marking category or others, can

be estimated by placing data-collection subprocedures in the appropriate places in the procedure. For the activity-marking variables defined in Chapter 3, one need only collect (probabilistic) information regarding the time spent in each marking and the number of completions of each activity. More general variables, that depend on sequences of reached markings and activity completions, can also be estimated. Details of how this is done are presented in the Appendix, where, among other things, an actual implementation of a simulation procedure similar to the one given here is discussed.

CHAPTER VI

APPLICATIONS

Introduction

To illustrate the applicability of stochastic activity networks to the modeling of realistic systems, two applications are considered. The first is a performability evaluation of an industrial computer network employing the IEEE 802.4 token bus protocol [39], where the performance of the network is subject to noise bursts and token losses. The study illustrates the use of simulation as a solution method. The evaluation results obtained show 1) that stochastic activity networks are an appropriate model type for evaluating the performability of local area networks and 2) that the IEEE 802.4 protocol is extremely tolerant to token losses and noise bursts under moderate network loading.

The second study is an evaluation of a CSMA/CD local area network. It both illustrates the state-space savings achieved through the use of reduced base model construction methods and investigates the effect that a particular priority scheme has on the delay messages experience when trying to access the channel.

Token Bus Network

When designing and evaluating computer networks for industrial applications, it is important to characterize the performance and reliability of the network in this context.

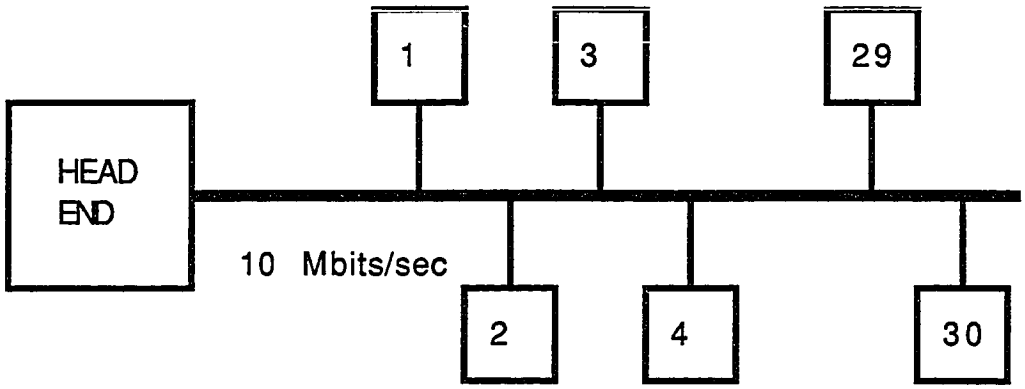


Figure 6.1: 30 Station Token Bus Network

For example, in manufacturing applications, it is important to determine if a given network topology and architecture will satisfy the real time requirements imposed by certain application processes. The Manufacturing Automation Protocol (MAP) [51] is one such protocol that has been proposed for these applications. Work to date concerning the performance evaluation of MAP and the IEEE 802.4 protocol has been widespread (see, for example [10,11,42,80]), but limited by the assumption of fault-free operation. While this assumption suffices for certain systems and environments, it is not realistic for networks operating in typically hostile manufacturing environments. Studies that account for performance in the presence of faults are important in this case because, in the event of failures, a communication network does not simply stop operating, but continues to operate at a degraded level of service.

This work is a continuation of that reported in [72], which considered a detailed stochastic activity network model of a single station. In the study that follows, we evaluate the performability of a 30-station token bus network shown in Figure 6.1. The protocol used in the network is the IEEE 802.4 token bus standard protocol which corresponds to the

physical and data link layers of the OSI reference model [39] and is a component of the MAP network architecture. The performance variables considered are the mean response time, the mean token rotation time, and the fraction of time various frames (data frames, claim frames, token frames, and solicit successor frames) are on the bus for a variety of workload, noise and token loss conditions.

Model Assumptions

Several assumptions are made in transforming the token bus network considered into its model representation. In particular, our evaluation presumes a static network structure of 30 nodes. This static structure dictates that 1) no station is allowed to leave the logical ring formed by token bus protocol, 2) no new stations are allowed to enter the ring, and 3) ring collapse is not allowed when the token is lost. Token loss is modeled in a probabilistic manner via a specified probability (a parameter of the model) of token loss during each pass of the token. Inter-arrival times of noise bursts are assumed to have a normal distribution; the duration of a burst is fixed at approximately 4 slot times.

In addition, several assumptions are made concerning the individual stations. Specifically, it is assumed that

1. Each station is up and operational (i.e., in the UP state), and will not be turned off during the course of simulation. Thus, it is not necessary to consider the OFF_LINE state.

2. All frames sent are of the type request_with_no_response (i.e., no acknowledgements are expected at this layer). This allows the state AWAIT_IFM_RESPONSE and CHECK_ACCESS_CLASS to be combined. This is in accordance with the MAP 2.1 network architecture (i.e. IEEE 802.2 with class I option).

3. Each protocol machine implements only the highest priority class (class 6).

4. Each station always knows its successor and its predecessor in the logical ring.

Several additional assumptions are made concerning the network. In particular

Slot_Time Based Timers		
Name	*Slot Time Units*	*Value in μs*
bus_idle_timer	49	224
response_window_timer	2	11.2
token_pass_timer	4	22.4

Octet Interval Timers		
Name	*Octet Time Units*	*Value in μs*
token_rotation_timer	187500	150000
token_hold_timer	5000	4000

Table 6.1: Timer Values for Network Model

1. Frame generation is modeled as a Poisson process. Frame size is fixed at 1024 bytes.

2. The data rate = 10 M bits/sec, Octet_interval = 0.8 μs, and Slot_time_interval = 5.6 μs.

Table 6.1 shows numerical values for the various timers, delays, and other network parameters.

Token Bus Network Model

The stochastic activity network model of the token bus protocol developed in this study is based on the reduced state transition diagram given in Figure 6.2. This specification was obtained from the standard access control machine (ACM) state transition specification [39], taking into account the assumptions made in the previous section. In particular, note that DEMAND_IN and DEMAND_DELAY need not be represented since the structure of the logical ring is static. Additionally, it is not necessary to consider the OFF_LINE state since it is assumed that no stations leave the ring voluntarily during the operation period. Finally, state AWAIT_IFM_RESPONSE and CHECK_ACCESS_CLASS can be combined since all data frames sent are of the type "request_with_no_response".

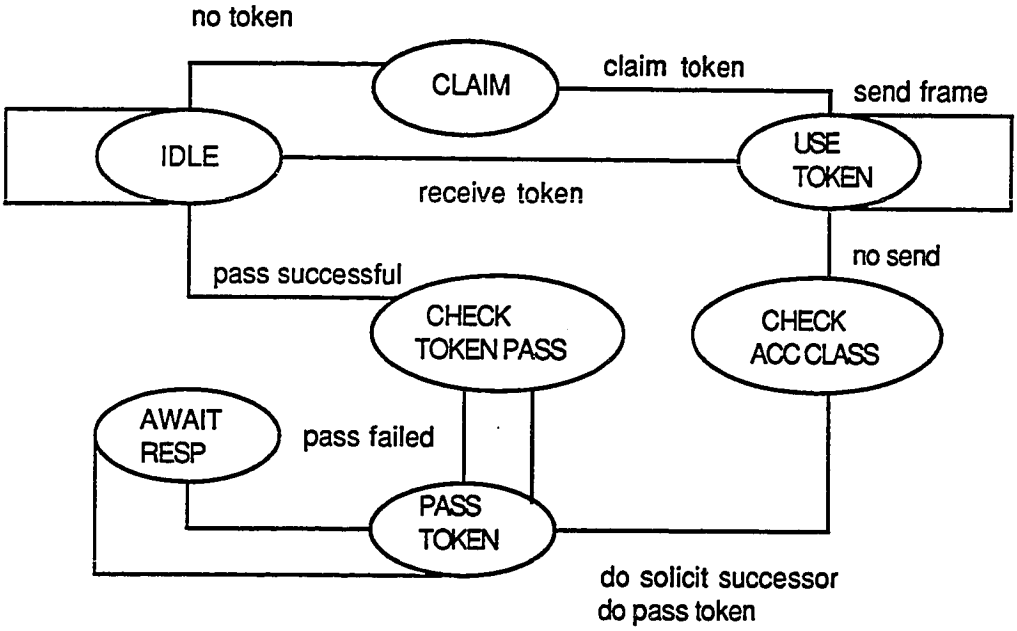


Figure 6.2: Reduced State Transition Diagram

Preconditions for a change in protocol state and postconditions after the state change are modeled by input gates and output gates respectively. Activity names are associated with protocol transition names (i.e., *receive_token*, *do_pass_token*). The token bus itself is represented by the place *pbus* in the model. The marking of this place indicates the activity taking place on the bus. For convenience, each possible marking of *pbus* is referred to by a textual label to aid in understanding node behavior. The markings for data frames, data frames corrupted by noise, claim frame, solicit successor frame, corrupted protocol frame, and token frame are thus *data_frame*, *corrupt_data*, *claim_frame*, *sol_succ_frame*, *corrupt_protocol*, and *token_frame* respectively. The marking representing an idle bus is *idle* and the markings representing token frames are the non-zero integer less than or equal to 30.

Given this bus representation, the model for the entire network is constructed by developing a representation for a single station and replicating it 30 times, holding the place *pbus* common among all the replicate station models. Communication between stations is thus only through the bus, as it is in a real network. A stochastic activity

Activity	Rate	Probability	
		case 1	case 2
tx_frame_gen	exp(λ)	1	-
data_frame_delay	determ(800)	1	-
start_trans1	inst	1	-
token_hold_timer	determ(4000)	1	-
do_solicit_successor	inst	1	-
do_pass_token	inst	1	-
token_rotation_timer	determ(150000)	1	-
no_send	determ(0.1)	1	-
response_window_timer	if (*station_id*==*low_station_id*) determ(11.2) else determ(5.6)	1	-
ss_failed	inst	1	-
solicit_succ_frame_delay	determ(20.8)	1	-
token_frame_delay	determ(20.8)	1	-
start_trans2	inst	1	-
pass_successful	inst	1	-
token_pass_timer	determ(22.4)	1	-
get_token	determ(.1)	1 - *prob_loss*	*prob_loss*
bus_idle_timer	determ(224)	1	-
start_trans3	inst	1	-
claim_token_delay	determ(750.0*(*station_id*/*high_station_id*))	1	-
start_trans4	inst	1	-

Table 6.2: Activity Parameters for Network Node

network model for each station is shown in Figure 6.3. In order to make the diagram more readable, several places (e.g., *pbus*) are pictured in several locations in the diagram, and are filled with a unique pattern to indicate that they actually represent a single place in the SAN. Activity, input gate, and output gate parameters are given in Tables 6.2, 6.3, and 6.4 respectively.

Token losses in the network are modeled by assuming that there is a certain probability that the token will be lost each time it is passed. This probability is represented by the cases of the activity *get_token* in Figure 6.3. A stochastic activity network model representing the generation of noise bursts is shown in Figure 6.4. Noise bursts are assumed to arrive in a normally distributed fashion. The effects of noise bursts on the token bus

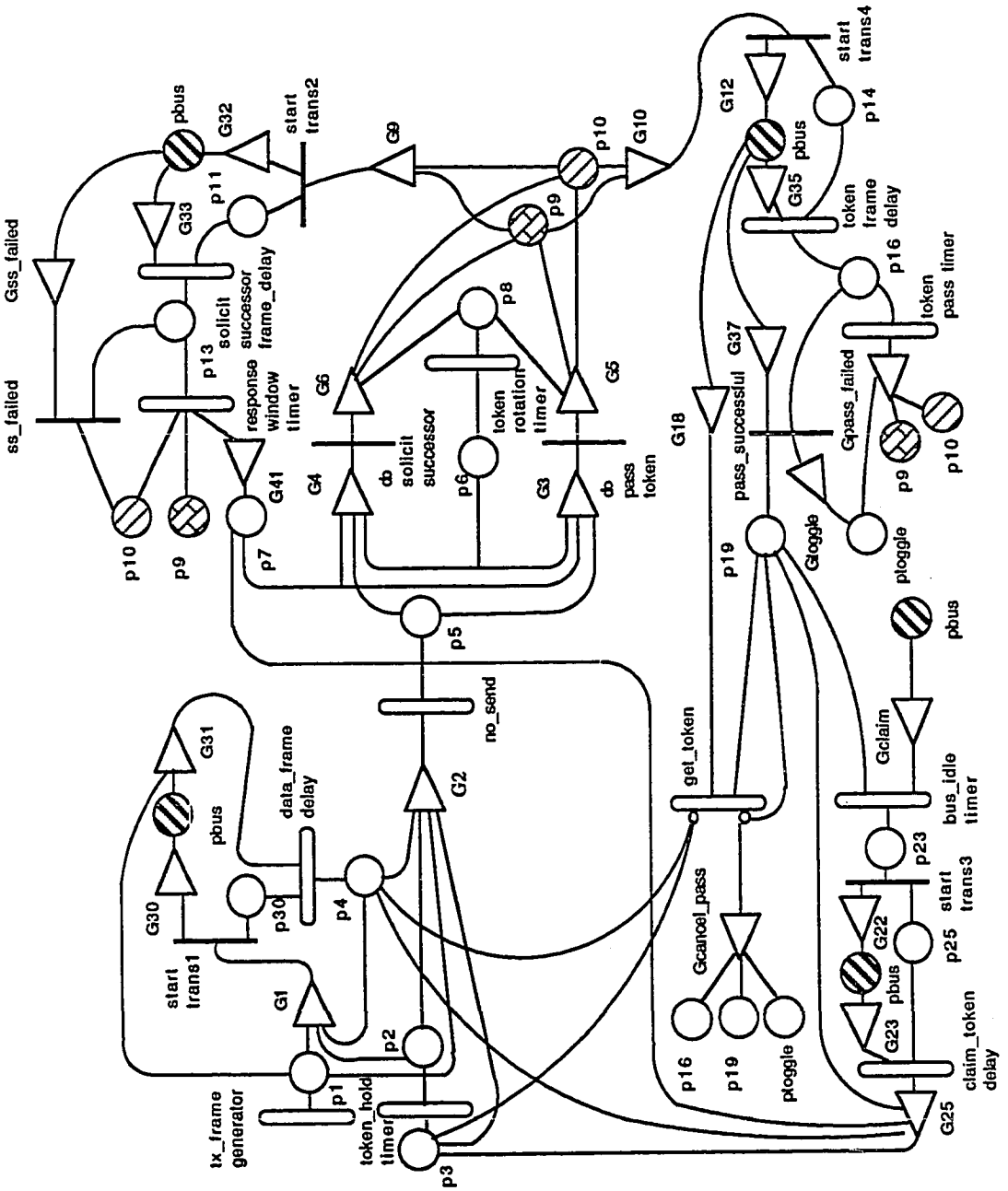


Figure 6.3: Station Model

Gate	Enabling Predicate	Function
$G1$	MARK($P1$)≥1 and MARK($P2$)==0 and MARK($P4$)≥1	MARK($P4$)=MARK($P4$)-1;
$G2$	(MARK($P2$)≥1 or MARK($P1$)==0) and MARK($P4$)≥1	if (MARK($P2$)≥1) MARK($P2$)=MARK($P2$)-1; MARK($P3$)=0; MARK($P4$)=MARK($P4$)-1;

$G3$	(MARK($P6$)==1 or MARK($P7$)>0) and MARK($P5$)≥1	MARK($P5$)=MARK($P5$)-1;
		if (MARK($P7$)≥1)
		MARK($P7$)=MARK($P7$)-1;
		if (MARK($P6$)≥1) MARK($P6$)=0;

$G4$	MARK($P5$)≥1 and MARK($P7$)==0 and MARK($P6$)==0	MARK($P5$)=MARK($P5$)-1; MARK($P6$)=0;
$G9$	MARK($P10$)==1 and MARK($P9$)==0	MARK($P10$)=0;
$G10$	MARK($P10$)==1 and MARK($P9$)==1	MARK($P10$)=0; MARK(P9)=0;
$G18$	*station_id*==MARK(*pbus*)	MARK(*pbus*)=*idle*;

$G23$	1	if (*station_id*==*high_station_id*)
		if (MARK(*pxx*)≠1)
		MARK(*pbus*)=*idle*;
		else MARK(*pbus*)=*corrupt_carrier*;

$G31$	1	if (MARK(*pbus*)==*corrupt_data*)
		if (MARK(*pxx*)≠1)
		MARK(*pbus*)=*idle*;
		else MARK(*pbus*)=*corrupt_carrier*;
		else
		MARK(*pbus*)=*idle*;
		MARK($P1$)=MARK($P1$)-1;

$G33$	1	if (MARK(*pxx*)≠1) MARK(*pbus*)=*idle*; else MARK(*pbus*)=*corrupt_carrier*;
$G35$	1	if (MARK(*pbus*)≠*corrupt_protocol*) MARK(*pbus*)=*next_station_id*; else if (MARK(*pxx*)≠1) MARK(*pbus*)=*idle*; else MARK(*pbus*)=*corrupt_carrier*;
$G37$	MARK(*pbus*)≠*next_station_id* and MARK(*pbus*)≠*idle* and MARK(*pbus*)≠*corrupt_protocol* and MARK(*pbus*)≠*corrupt_carrier*	identity
Gss_failed	MARK(*pbus*)==*corrupt_carrier* or MARK(*pbus*)==*corrupt_protocol*	identity
Gclaim	MARK(*pbus*)==*idle*)	identity

Table 6.3: Input Gate Parameters for Network Node

Gate	Function
$G5$	MARK($P8$)=1; MARK($P9$)=1; MARK($P10$)=1;
$G6$	MARK($P9$)=0; MARK($P10$)=1; MARK($P8$)=1;
$G12$	if (MARK(pxx)≠1) MARK($pbus$)=*token_frame*; else MARK($pbus$)=*corrupt_protocol*;

$G22$	if (*station_id*==*high_station_id*)
	if (MARK(pxx)≠1)
	MARK($pbus$)=*claim_frame*;
	else MARK($pbus$)=*corrupt_CF*;

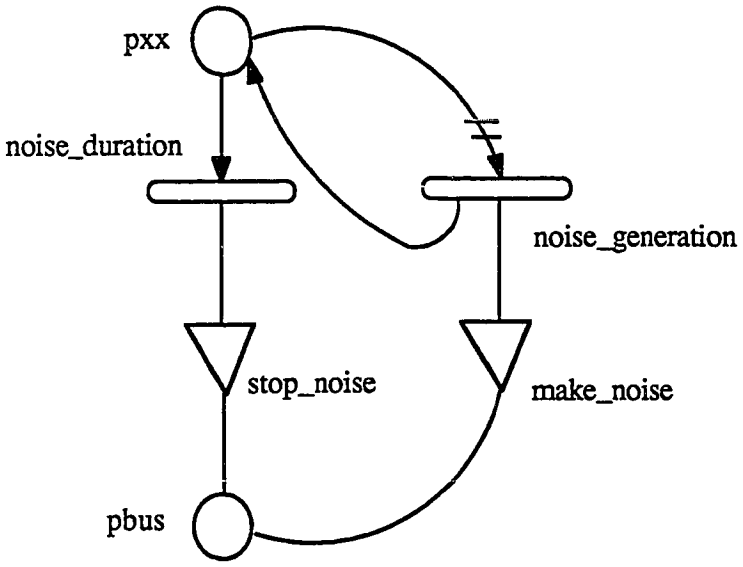
$G25$	if (*station_id*==*high_station_id*)
	MARK($P4$)=1; MARK($P3$)=1;
	MARK($P7$)=20;
	else
	MARK($P19$)=1; MARK($P7$)=20;

$G30$	if (MARK(pxx)≠1) MARK($pbus$)=*data_frame*; else MARK($pbus$)=*corrupt_data*;
$G32$	if (MARK(pxx)≠1) MARK($pbus$)=*sol_succ_frame*; else MARK($pbus$)=*corrupt_protocol*;
$G41$	MARK($P7$)=20;
Gcancel_pass	MARK($P16$*_previous_station*)=0; MARK($P19$*_previous_station*)=1; MARK(*Ptoggle_previous_station*)=0;

Gpass_failed	if (MARK(*Ptoggle*)==0)
	MARK(*Ptoggle*)=1; MARK($P9$)=1;
	MARK($P10$)=1;
	else
	MARK(*Ptoggle*)=0; MARK($P9$)=0;
	MARK($P10$)=1;

Gtoggle	MARK(*Ptoggle*)=1;

Table 6.4: Output Gate Parameters for Network Node



Gate	Function
stop_noise	if (MARK(*pbus*)==*corrupted_carrier*) MARK(*pbus*)=*idle*; else if (MARK(*pbus*)==*corrupted_CF*) MARK(*pbus*)=*claim_frame*;
make_noise	if (MARK(*pbus*)==*data_frame*) MARK(*pbus*)=*corrupted_data*; else if (MARK(*pbus*)=*idle*) MARK(*pbus*)=*corrupted_carrier*; else if (MARK(*pbus*)=*claim_frame*) MARK(*pbus*)=*corrupted_CF*; else MARK(*pbus*)=*corrupted_protocol*;

Activity	Distribution
noise_generation	normal(*time_between_noise*, *time_betwween_noise* * .2)
noise_duration	determ(22.4)

Figure 6.4: Noise Burst Model

network are two-fold. If a noise burst affects a data frame, that data frame is retransmitted and the effects are recorded by the corrupted data on the bus (marking of place *pbus* is *corrupt_data*). If a noise burst affects any protocol frame, actions specified in the IEEE specification are taken and the effects are recorded by the corrupted protocol on the bus (marking of place *pbus* is *corrupt_protocol*).

Model Execution

Changes in the ACM state during network operation are represented by activity completions in the model and places represent the complete state of each network node. For example, when a station receives the token from its previous station or generates a token in the case of a token loss, the markings of places *P3* and *P4* are set to one. A marking of one in place *P3* enables the activity *token_hold_timer* which has an activity time equal to the value of 5000 octets (see Table 6.1). The marking of place *P1* indicates the number of outstanding packets that are queued at that station. Gate *G1* holds if the markings of places *P1* and *P3* are non-zero and the marking of place *P2* is zero (the marking of place *P2* indicates that *token_hold_timer* has not expired). This allows for the transmission of a packet queued at the station. The start of transmission is signaled by the completion of the activity *start_trans1* which changes the marking of place *pbus* to *data_frame*. The time for transmitting a packet is given by the activity time of the timed activity *data_frame_delay*. Packets are transmitted until there are none to send or until the *token_hold_timer* expires (i.e. activity *token_hold_timer* completes). When transmission ceases, the predicate for gate *G2* is satisfied, enabling the instantaneous activity *no_send*.

At this point, one of two things can happen, depending of the **maximum inter-solicit count**, as represented by place *P7*. If the marking of *P7* is not zero, the token is passed, and if it is zero, successors are solicited. The token is passed by enabling the activity *do_pass_token* and successors are solicited by enabling the activity *do_solicit_successor*. In the case of solicit successor, the predicate of gate *G9* is satisfied enabling activity *start_trans2*. Solicit successor frames on the bus are indicated by a marking of *sol_succ_frame* in the place *pbus*. Once the solicit successor is successful, the token is passed. Upon completion of either a solicit successor operation or activity *do_pass_token*,

Activity Name	*Distribution*	*Parameters in μs*
tx_frame_generator	exponential	varied
data_frame_delay	deterministic	800.0
solicit_succ_frame_delay	deterministic	20.8
token_frame_delay	deterministic	20.8
get_token	deterministic	0.1
claim_token_delay	deterministic	dependent on station ID
noise_generation	normal	varied
noise_duration	deterministic	22.4

Table 6.5: Timed Activity Distributions for Network Model

the predicate for gate *G10* is satisfied and a token frame is placed on the bus. This action enables the timed activity *token_frame_delay*. The activity time of *token_frame_delay* is equivalent to the token transmission time interval. If the token pass fails, the token pass is tried a second time and if it fails again, a solicit successor is done to find the next station to which the token is passed.

In the case of a token loss, signified by the completion of the timed activity *bus_idle_timer*, the instantaneous activity *start_trans3* is enabled to start a claim token phase. Claim token frames are indicated by a marking of *claim_frame* in the place *pbus*. Claim frames are sent for a period whose length is a function of station ID. At the completion of the activity *claim_frame_delay* a token is generated and the network resumes normal operation.

Model execution thus mimics actual protocol operation. A change in protocol state, an expiration of a timer, a frame arrival, etc., all cause a change in the marking of the network. By observing the execution of the stochastic activity network model one can infer the protocol behavior. Table 6.5 lists the timed activities associated with the model with their corresponding distributions and parameter values.

Performance Variables

The performance variables studied include the mean response time, the mean token rotation time, and the fraction of time each type of frame (data or protocol) is on the

bus, for a variety of workload and environmental conditions. Nine types of bus activities are considered, corresponding to the different data and protocol frames that can be on the bus together with possible error conditions. Each of these variables is formulated as the fraction of time that the particular condition existed in steady-state. Specifically, the following types of bus activity are considered: data transmission, idle bus, token on the bus, solicit successor frame present, corrupted protocol frame present, corrupt carrier present, claim frame present, corrupted data present, listen for solicit successor. The name used for each variable in the input code and reporting of the results is, respectively, *data_on_bus*, *idle_bus*, *token_on_bus*, *soll_succ_frame*, *corrupt_pro*, *corrupt_carrier*, *claim_frame*, *cdata_on_bus*, and *listen_soll_succ*.

Response time characteristics of the network under a variety of conditions are also evaluated. In this case, the mean number of packets at a node in steady-state (including the one in transmission, if any) is estimated and Little's result is used to obtain the expected combined queueing and transmission time. Response times for several different stations are studied to see whether position in the logical ring was a significant factor. For the remainder of this study, the station next in the ring after the *high station* is referred to as the *low station*, and the station equally distant from each of these stations, logically, is referred to as the *middle station*. Response times for the low and high stations are estimated for all conditions studied. Response times for the middle station are estimated only for high load conditions. In the results section, the variables *num_in_sysHS*, *num_in_sysLS*, and *num_in_sysMS* refer to the expected number of packets at each station (in steady-state) for the high, low, and middle stations respectively. Expected response times are then computed by dividing these estimates by the chosen arrival rate at the station. In addition, the expected token rotation time is estimated for each experiment. In the tables given in next section, this variable is referred to as *token_rotation*.

Discussion of Results

Three sets of simulation experiments were conducted to evaluate the performability of the token bus network. The resulting performance variable values are given in Tables 6.6 to 6.10. All intervals reported in the tables are at a 90% confidence level.

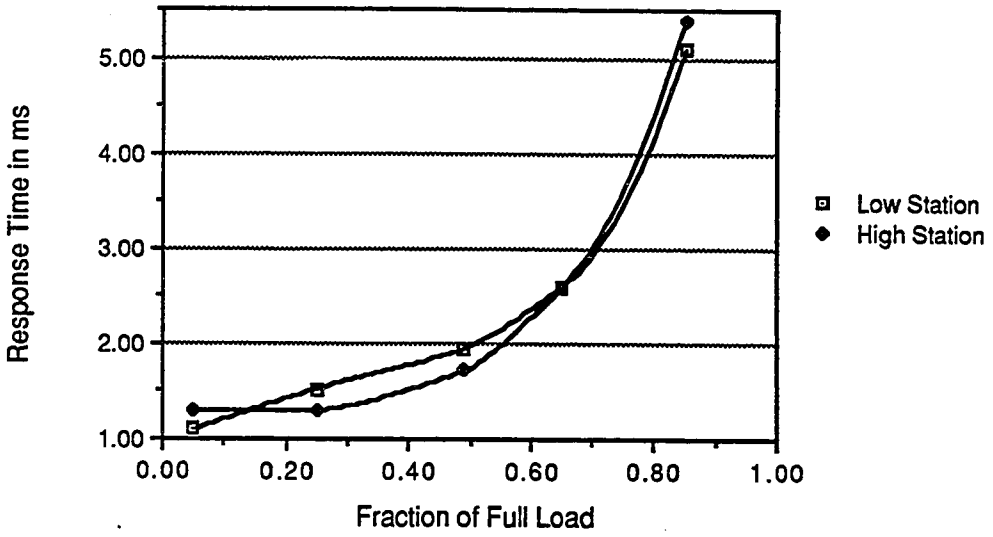


Figure 6.5: Response Time vs. Load

(Token Loss Prob. .001, Mean Time Between Noise Bursts 400 ms)

In the first set of experiments, network behavior was studied under varying arrival rate of data packets to the station (5% to 85% of bus capacity and a Poisson arrival process). Resulting values of the performance variables are given in Table 6.6. Figure 6.5 is the response time at the high station and the low station. The response time is the sum of queueing delay and the frame transmission time (in this case transmission time is a constant 800 μs).

The response time behavior was analyzed with a token loss probability of 0.001 and a noise burst rate with an expected time between bursts of 400 milliseconds as the load on the network was varied. The response time behavior for a 30 station model, even at 85% load, is good and has an average value of 5.25 milliseconds. However, from Figure 6.5, the operating point for the token bus network is the knee portion of the curve shown and is at about 60% load. Though Figure 6.5 indicates different values for response time at the low

Time Between Noise Bursts 400 ms, Token Loss Prob. .001				
Fraction of Full Load	data_on_bus (frac. of time)	idle_bus (frac. of time)	token_on_bus (frac. of time)	soll_succ_frame (frac. of time)
.05	.051 ± .002	.0228 ± .0004	.857 ± .003	.0331 ± .0006
.25	.250 ± .007	.0177 ± .0004	.679 ± .007	.0264 ± .0007
.50	.494 ± .009	.012 ± .0004	.457 ± .008	.018 ± .0007
.65	.646 ± .009	.0085 ± .0031	.319 ± .008	.0122 ± .0005
.85	.847 ± .006	.0037 ± .0002	.137 ± .005	.0052 ± .005

Time Between Noise Bursts 400 ms, Token Loss Prob. .001				
Fraction of Full Load	corrupt_pro (frac. of time)	corrupt_carrier (frac. of time)	claim_frame (frac. of time)	cdata_on_bus (frac. of time)
.05	.000024 ± .000006	.00003 ± .000006	.031 ± .002	.00009 ± .00008
.25	.000023 ± .000008	.000020 ± .000007	.023 ± .001	.0002 ± .0001
.50	.000016 ± .000009	.000014 ± .000007	.016 ± .002	.0006 ± .0002
.65	.000009 ± .000005	.000009 ± .000004	.012 ± .001	.00007 ± .00002
.85	.000009 ± .000003	.000007 ± .000003	.0052 ± .0005	.0009 ± .0002

Time Between Noise Bursts 400 ms, Token Loss Prob. .001			
Fraction of Full Load	num_in_sys HS	num_in_sys LS	token_rotation (ms)
.05	.0026 ± .0007	.0022 ± .0007	.718 ± .002
.25	.013 ± .002	.016 ± .002	.90 ± .11
.50	.035 ± .004	.039 ± .005	1.37 ± .07
.65	.062 ± .007	.061 ± .006	1.96 ± .07
.85	.19 ± .01	.18 ± .01	4.6 ± .6

Table 6.6: Experiments with Load Varied

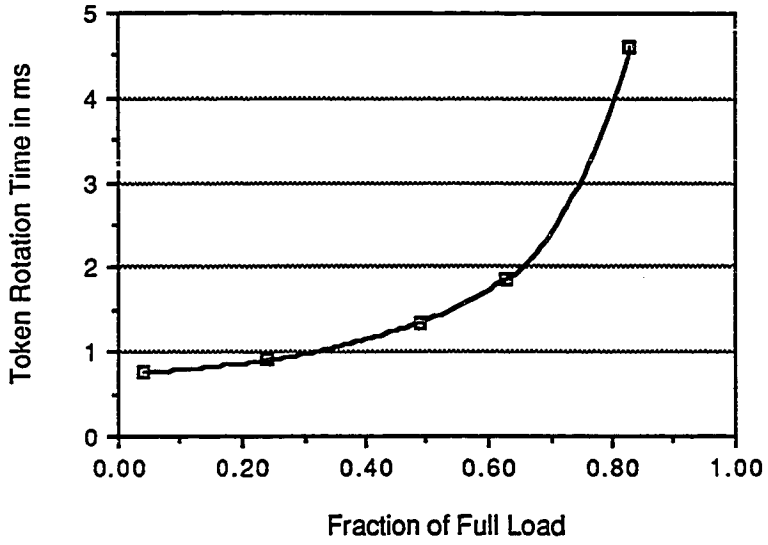


Figure 6.6: Token Rotation Time vs. Load

(Token Loss Prob. .001, Mean Time Between Noise Bursts 400 ms)

station and the high station, response times are not statistically different. This response time behavior is attributed to the increase in queueing delays as the load is increased. It should be noted also that the high priority token hold time for each station in the network was assumed to have the same value of 4 milliseconds.

The token rotation time for the network is shown in Figure 6.6. The token rotation time shown in Figure 6.6 increases with increase in load and has an average value of 4.5 milliseconds even at 85% load. This behavior is due to the large value of high priority token hold time used for the stations. It indicates that the network was not saturated at this combination of high priority token hold time, packet size, and load conditions.

The behavior of other performance variables is shown as bus activity in Figure 6.7. These include: data frames, idle bus, token frames, solicit successor frames, claim frames, and corrupt data and protocol frames. This behavior is shown to indicate that the model developed for the network closely follows the expected behavior of the token bus network. As depicted in Figure 6.7, when the load increases, the percentage of time that the data

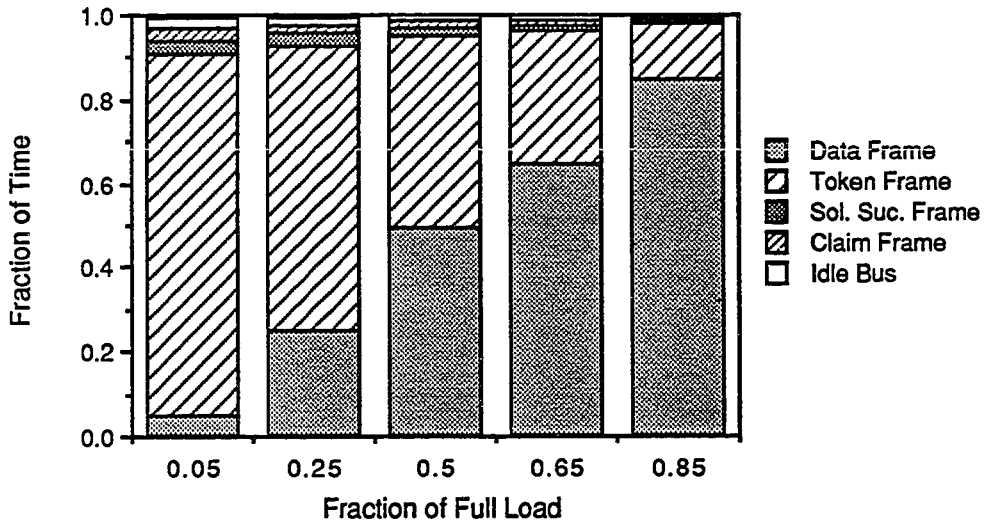


Figure 6.7: Bus Activity in Steady-State

(Token Loss Prob. .001, Mean Time Between Noise Bursts 400 ms)

50% Full Load, Token Loss Prob. .001			
Time Between Noise Bursts (ms)	num_in_sys HS	num_in_sys LS	token_rotation (ms)
10	.042 ± .004	.047 ± .005	1.46 ± .3
20	.041 ± .004	.037 ± .004	1.38 ± .2
30	.036 ± .004	.042 ± .005	1.38 ± .3
40	.037 ± .004	.045 ± .005	1.39 ± .4
130	.038 ± .004	.037 ± .004	1.36 ± .2
220	.037 ± .003	.038 ± .003	1.34 ± .15
310	.039 ± .003	.038 ± .003	1.34 ± .15
400	.035 ± .004	.039 ± .005	1.37 ± .07

Table 6.7: Experiments with Noise Burst Rate Varied, 50% Load

frames are on the bus increases and that of other frames decreases.

In the second set of experiments, the network behavior was studied with 50% and 85% load and varying noise burst rates (expected time between noise bursts was 10 to 400 ms). Resulting values of the performance variables are given in Tables 6.7 and 6.8.

The response time behavior at the high station and the low station under varying noise burst rates is shown in Figure 6.8. It is interesting to note that token bus network behavior is relatively immune to noise bursts. This is further emphasized by the fact that the experiments were conducted under extremely high noise burst rates (expected time between noise bursts was 10 to 400 ms). The average value of the response time under the condition of 50% load and high noise burst rate is 2.2 ms and is statistically very close to that with low noise burst rate (2 ms).

The behavior of the network under 85% load and varying noise burst rate is also shown in Figure 6.8. In Figure 6.8, the response times at the low station, the middle station, and the high station are shown for an 85% load. Figure 6.8 depicts an increase in response time at high noise burst rates. This is attributed to the increase in chance of noise burst affecting the data frames. However, it should be noted that the token bus network generally exhibits high immunity to noise bursts, which is extremely important in the context of plant floor applications. The average value of the response at 85% load

85% Full Load, Token Loss Prob. .001				
Time Between Noise Bursts (ms)	num_in_sys HS	num_in_sys LS	num_in_sys MS	token_rotation (ms)
10	.39 ± .03	.39 ± .04	.37 ± .03	9.5 ± .9
30	.23 ± .02	.24 ± .02	.23 ± .02	5.6 ± .3
130	.17 ± .02	.20 ± .02	.19 ± .02	4.6 ± .3
310	.17 ± .02	.20 ± .02	.20 ± .02	4.4 ± .3
400	.19 ± .01	.18 ± .01	.19 ± .01	4.6 ± .6

Table 6.8: Experiments with Noise Burst Rate Varied, 85% Load

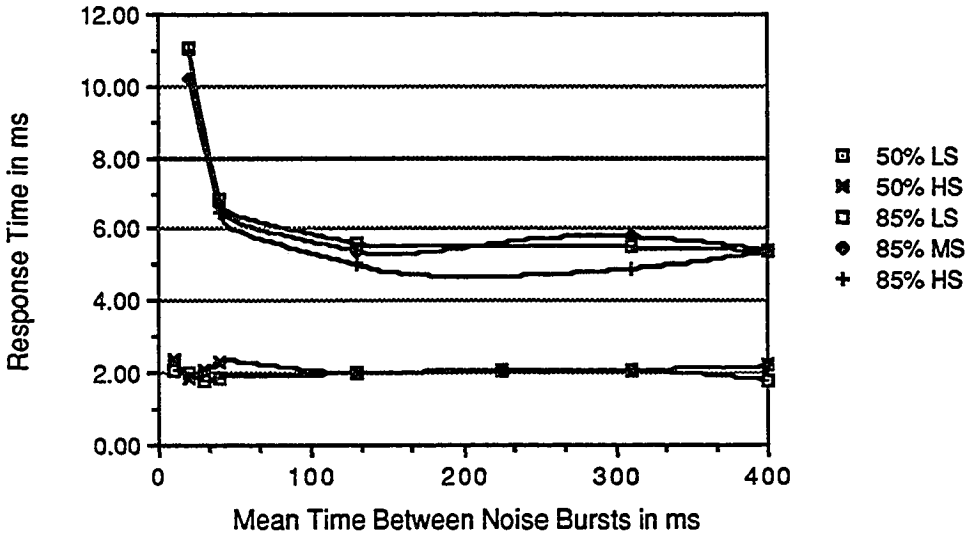


Figure 6.8: Response Time vs. Noise Condition

(Data Transmission Time .8 ms)

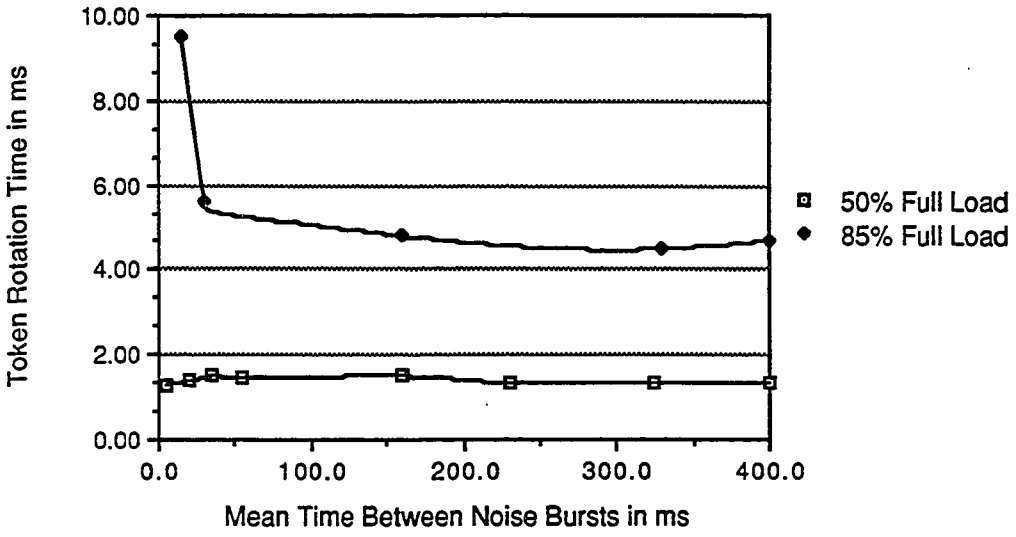


Figure 6.9: Token Rotation Time vs. Noise Condition

(Data Transmission Time .8 ms)

and high noise burst rate is 11 ms.

The token rotation time for the network under 50% and 85% load and varying noise burst rate is shown in Figure 6.9. It should be noted that there is no appreciable difference in the token rotation time behavior for the network under 50% load and low noise burst rate (expected time between noise bursts was 400 ms) and that for 50% load and high noise burst rate (expected time between noise bursts was 10 ms). The token rotation time is doubled with 85% load and high noise burst rate when compared to that at 85% load and low noise burst rate.

In the third set of experiments, the network behavior was studied under 50% and 85% load and varying token loss probability (0.001 to 0.1). (Note that a token loss probability of 0.1 implies that on the average one token is lost for every 10 token passes. This represents an extremely severe condition in the network.) The token loss probability indicates the probability that a station will lose the token when passing it to the next station. Further, it is assumed that this token loss will not bring down the entire logical ring. Resulting

50% Full Load, Mean Time Between Noise Bursts 400 ms			
Prob. of Token Loss per Pass	num_in_sys HS	num_in_sys LS	token_rotation (ms)
.001	.035 ± .004	.039 ± .005	1.37 ± .07
.005	.038 ± .003	.040 ± .003	1.46 ± .01
.01	.040 ± .003	.042 ± .003	1.6 ± .2
.03	.044 ± .005	.051 ± .004	2.1 ± .3
.06	.051 ± .004	.054 ± .004	2.3 ± .35
.1	.069 ± .007	.066 ± .007	2.5 ± .3

Table 6.9: Experiments with Token Loss Probability Varied, 50% Load

85% Full Load, Mean Time Between Noise Bursts 400 ms				
Prob. of Token Loss per Pass	num_in_sys HS	num_in_sys LS	num_in_sys MS	token_rotation (ms)
.001	.19 ± .01	.18 ± .01	.19 ± .01	4.6 ± .6
.005	.202 ± .026	.194 ± .025	.213 ± .025	5.1 ± .3
.01	.217 ± .035	.24 ± .03	.23 ± .04	5.7 ± .5
.03	.23 ± .02	.24 ± .02	.30 ± .02	7.2 ± .4
.06	.33 ± .02	.35 ± .02	.57 ± .04	8.5 ± .4
.1	.47 ± .05	.54 ± .06	.97 ± .09	8.3 ± .5

Table 6.10: Experiments with Token Loss Probability Varied, 85% Load

values of performance variables are given in Tables 6.9 and 6.10. The response time behavior at the high station and the low station is shown in Figure 6.10. Though the response times differ for the high and the low stations, they are not statistically different because of their confidence intervals. The response time of the token bus network increases from 2 ms to 3 ms at 50% load and token loss probability varying from 0.001 to 0.1.

The response times at the low station, the middle station, and the high station under an 85% load are shown in Figure 6.10. Note that the response time increases with increase in token loss probability. The response time for the middle station increases from 4.5 ms with 0.005 token loss probability to 27 ms with 0.1 token loss probability.

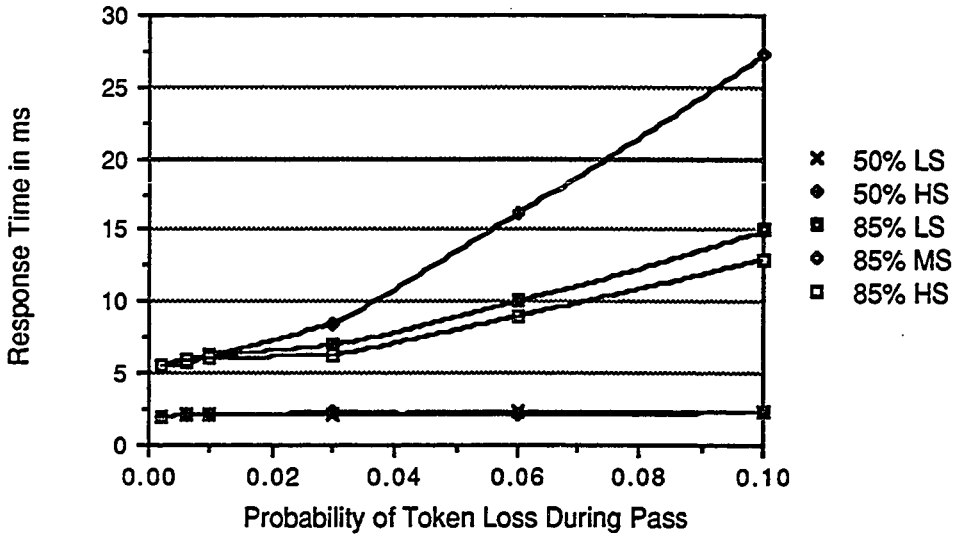


Figure 6.10: Response Time vs. Token Loss Probability

(Data Transmission Time .8 ms)

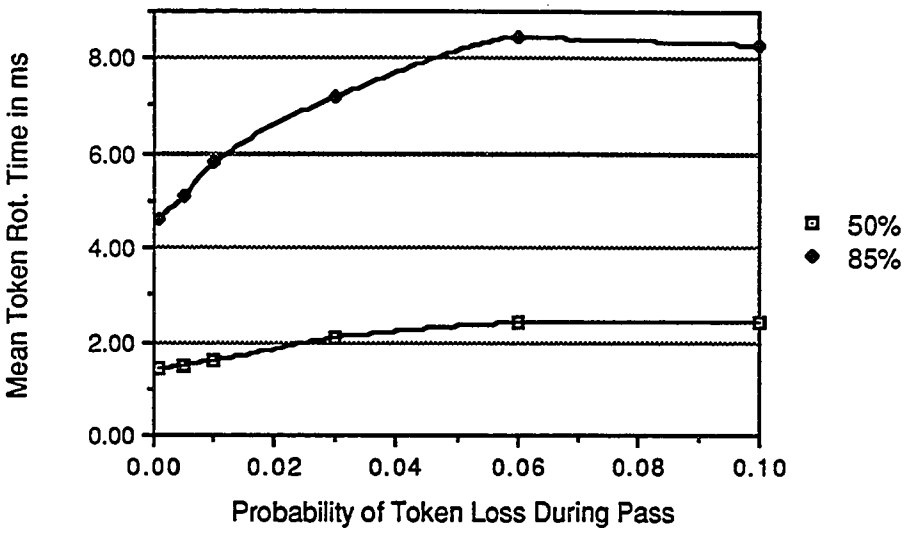


Figure 6.11: Token Rotation Time vs. Token Loss Probability

(Data Transmission Time .8 ms)

The token rotation time for the network as a function of the token loss probability is shown in Figure 6.11. At 50% load, one notes relatively little influence due to changes in token loss probability. Indeed, when there is a very high loss probability (0.1), the token rotation time still has a reasonable value (2.5 ms). The token rotation time behavior at 85% load is shown in Figure 11. The token rotation time increases from 4.5 ms to 8 ms when the token loss probability is varied from 0.005 to 0.1. Again, the token bus network performs relatively well even under severe token loss rates.

Conclusions

Hostile environment conditions persisting in factory floors result in noise bursts and token losses which must be accounted for when modeling the network. Stochastic activity networks and performability are appropriate for studying networks of this type which, in the presence of failures, continue to operate at a degraded level of service. Failure effects are taken into account at the SAN model level in an explicit and a systematic manner.

The results obtained in this study provide new and important information regarding the behavior of IEEE 802.4 token bus networks. Noise bursts and random token losses have been shown to have minimal impact on network performance in the case moderate (50%) network loading. In addition, it has been shown that even severe noise bursts and token losses have a manageable impact on token bus networks operating at very high (85%) loading.

CSMA/CD Local Area Network

The second study is an evaluation of a CSMA/CD local area network. The purpose of this study is two-fold: to illustrate the state-space savings achieved through the use of reduced base model construction methods, and to investigate the effect of a particular priority scheme on the delay that messages experience when trying to access the network channel. The protocol employed is a variant of non-persistent CSMA/CD. When a station has something to send, it waits an exponentially distributed amount of time before attempting the transmission. After this delay, it senses the channel. If the station detects an idle channel, it begins to transmit. One of two events will then occur: either 1) the channel was actually idle, so the transmission begins normally, or 2) a collision occurs, since another station had begun transmitting, but its signal had not propagated to the first station at the time the channel was sensed. If a collision occurs, it is cleared after an exponentially distributed amount of time. If the station detects a busy channel, it returns to the wait state and after the delay period once again attempts to gain access to the channel.

Two scenarios are considered. In the first, all stations in the network are identical. In the second, one station is considered to be a high-priority station, and all other stations are considered to be normal priority. Here the priority of a station is determined by the mean time it waits before trying to access the bus; the high-priority station waits (on the average) one-tenth the time of the other stations. A variety of performance variables are considered. Specifically, we determine the average queue length at each node (normal and high priority) and the fraction of time the channel is: idle, transmitting an unpropagated message, transmitting a propagated message, or clearing a collision.

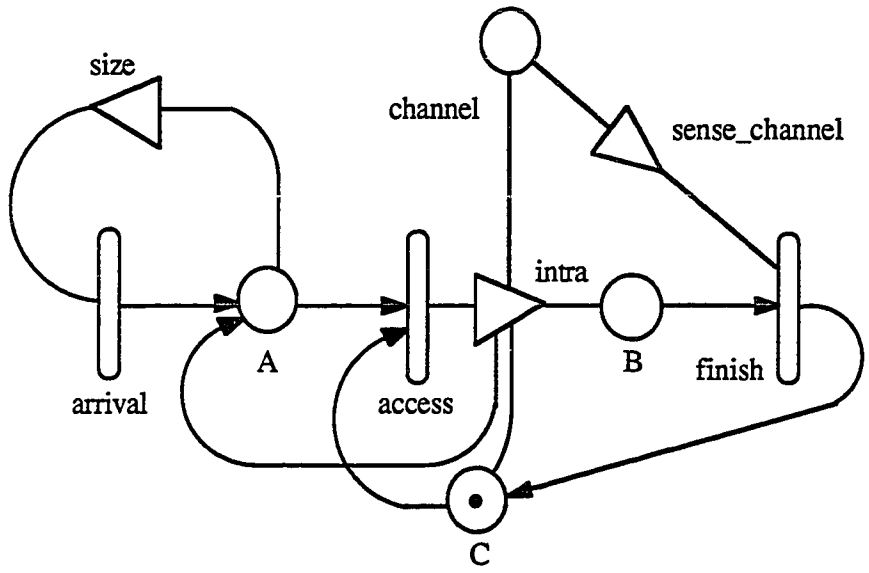


Figure 6.12: Station Submodel

Stochastic Activity Network Model and Variables

A stochastic activity network representing a single station connected to the channel is given in Figure 6.12. The gate and activity parameters for the model are given in Tables 6.11 and 6.12, respectively. The station represents either a high or normal priority station, depending on the duration of the expected wait between attempts to access the channel. Each completion of activity *arrival* represents the arrival of a message to the station queue. The station queue is represented by place A. The marking of place A represents the number of messages waiting to be transmitted. The finiteness of the queue is represented by gate *size* . Gate *size* specifies (via its predicate) that activity *arrival* is enabled only when the number of messages in the queue is less than the system's capacity.

Completion of activity *access* signals that the station is sensing the channel. This results in one of four outcomes, as stated above, which are implemented by gate *intra*. If the channel is idle (i.e. the marking of *channel* is 0), the marking of place *channel* is set to one to indicate the start of a transmission. In addition, the marking of B is set to one to indicate that a transmission is in progress for the station. If the channel is in use,

Gate	Type	Enabling Predicate	Function
size	input	$MARK(A) < 2$	identity

intra	output	-	if (MARK(*channel*)==0) {
			MARK(*channel*) = 1;
			MARK(*B*) = 1; }
			else if (MARK(*channel*)==1) {
			MARK(*channel*) = 3;
			MARK(*C*) = 1;
			MARK(*A*) = MARK(*A*) + 1; }
			else if (MARK(*channel*)==2) {
			MARK(*C*) = 1;
			MARK(*A*) = MARK(*A*) + 1; }
			else if (MARK(*channel*)==3) {
			MARK(*A*) = 1;
			MARK(*A*) = MARK(*A*) + 1; }

sense_channel	input	MARK(*channel*)==2) || MARK(*channel*)==3	MARK(*channel*) = 0;

Table 6.11: Gates for Station Submodel

Activity	Distribution Type	Parameter (Rate)
arrival	exponential	varied
access	exponential	10 (normal prio. station) 100 (high prio. station)

finish	exponential	if (MARK(*channel*)==2)
		rate = 1
		else
		rate = 5

Table 6.12: Activity Parameters for Station Submodel

but the signal has not received sufficient time to propagate (i.e. the marking of *channel* is 1), the marking of place *channel* is set to three to indicate that a corrupted message is now on the channel and a token is added to place A to indicate that a retransmission is necessary. If a corrupted message which has not been cleared is present (i.e. the marking of *channel* is 3), a token is added to place A to indicate that a retransmission is necessary and the station goes back into the wait state. If the transmission of a message that has been active long enough to propagate to all stations is in progress (i.e. the marking of *channel* is 2), a token is returned to place A to indicate that no transmission is possible at this time and the marking of place C is set to one to indicate that that station is again entering the wait state.

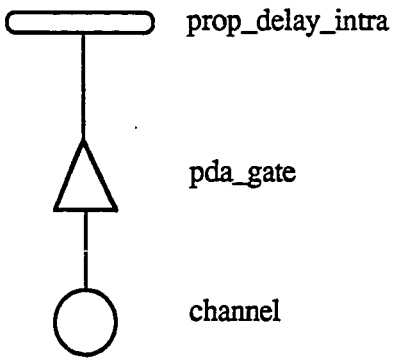
The time required to transmit messages (either corrupted or not) is controlled by activity *finish*, whose activity time distribution depends on the marking of place *channel*. This distribution depends on whether a propagated or corrupted message is on the channel. If a propagated message is on the channel, the distribution reflects the message transmission time. If a corrupted message is on the channel (detected by gate *sense_channel*), the distribution reflects the time to clear the collision. The completion of a transmission occurs when activity *finish* completes. When this happens, a token is added to place C.

The time to propagate a message is represented by the activity *prop_delay_intra* in the network submodel. The network submodel is shown in Figure 6.13. Gate *pda_gate* senses the presence of an unpropagated message (via its predicate) and, when it holds, activates activity *prop_delay_intra*. After a prescribed (exponentially distributed) propagation delay, activity *prop_delay_intra* completes, signaling that the propagation of the signal to all stations has occurred. The subsequent execution of gate *pd_gate* sets the marking of place *channel* to 2 thus indicating the presence of a propagated message.

Reward structures for the various performance variables are now considered. The first performance variable studied is the expected queue length in steady-state at each type of station. This can be obtained using the reward structure

$$C_{st}(a) = 0 \quad \forall a \in A$$

$$\mathcal{R}_{st}(\nu) = \begin{cases} i & \text{if } \nu = \{(A, i)\} \\ 0 & \text{otherwise} \end{cases}$$



Gate	Type	Enabling Predicate	Function
pda_gate	input	MARK(*channel*)=1	MARK(*channel*)=2;

Activity	Distribution Type	Parameter (Rate)
prop_delay_intra	exponential	20

Figure 6.13: Network Submodel

and variable $E[V_{t\to\infty}]$. For the homogeneous system, the expected queue length at an individual station is then $\frac{E[V_{t\to\infty}]}{N}$, where N is the number of stations of the given type. Since markings and activity completions in the network submodel do not contribute to this variable, the reward structure for this submodel $(\mathcal{C}_Z, \mathcal{R}_Z)$ is defined such that $\mathcal{C}_Z(a) = 0,\ \forall\ a \in A$, and $\mathcal{R}_Z(\nu) = 0,\ \forall \nu$.

Performance variables representing the fractions of time various actions are present on the channel are constructed using a null reward structure for each station submodel (i.e, $(\mathcal{C}_Z, \mathcal{R}_Z)$) and a reward structure for the network submodel such that:

$$\mathcal{C}_N(a) = 0 \ \ \forall a \in A$$

$$\mathcal{R}_N(\nu) = \begin{cases} 1 & \text{if } \nu = \{(channel, i)\} \\ 0 & \text{otherwise} \end{cases}$$

where i is the marking of the channel when the action is taking place and $E[V_{t\to\infty}]$ is the variable. For example, to determine the fraction of time a propagated packet is on the channel, i would be taken to be 2. The fractions of time that the bus is idle, an unpropagated message is on the bus, and a corrupted message is on the bus, are determined in a similar manner.

SAN-based reward models for the entire network are then constructed by replicating the normal and high-priority stations the required number of times and adjoining them with the network submodel and reward structure. The particular reward structures that are used depend on the particular performance variable being solved for. For example, to determine the expected queue length at a station for the homogeneous system (scenario 1), the SAN-based reward model given in Figure 6.14 can be used. In this figure, $S_{STATION}$ is the station submodel, $S_{NETWORK}$ is the network submodel, and n is the number of stations in the network. Similarly, if we define S_{HI} to be a high-priority station submodel, then a SAN-based reward model which can be used to determine the expected queue length at the high-priority station is given in Figure 6.15, where n is now the number of normal priority stations in the model. SAN-based reward models for the other variables can be constructed in a similar manner.

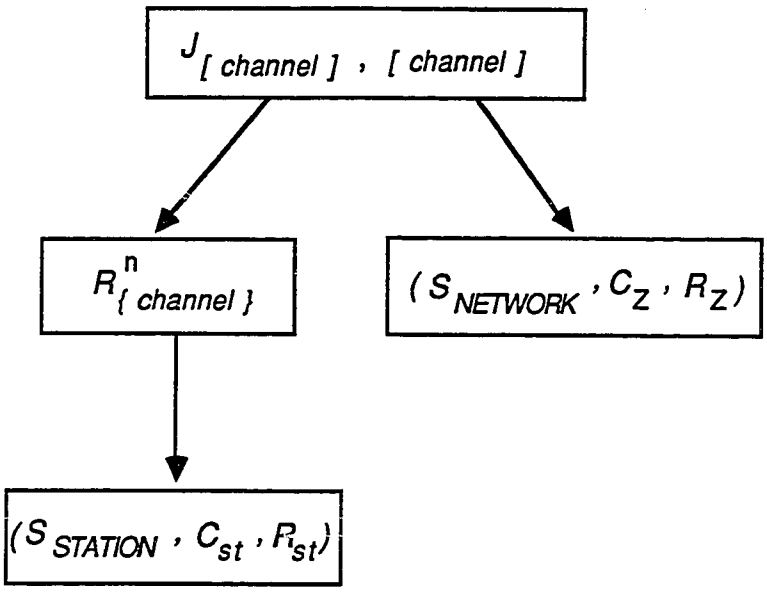


Figure 6.14: Homogeneous Network

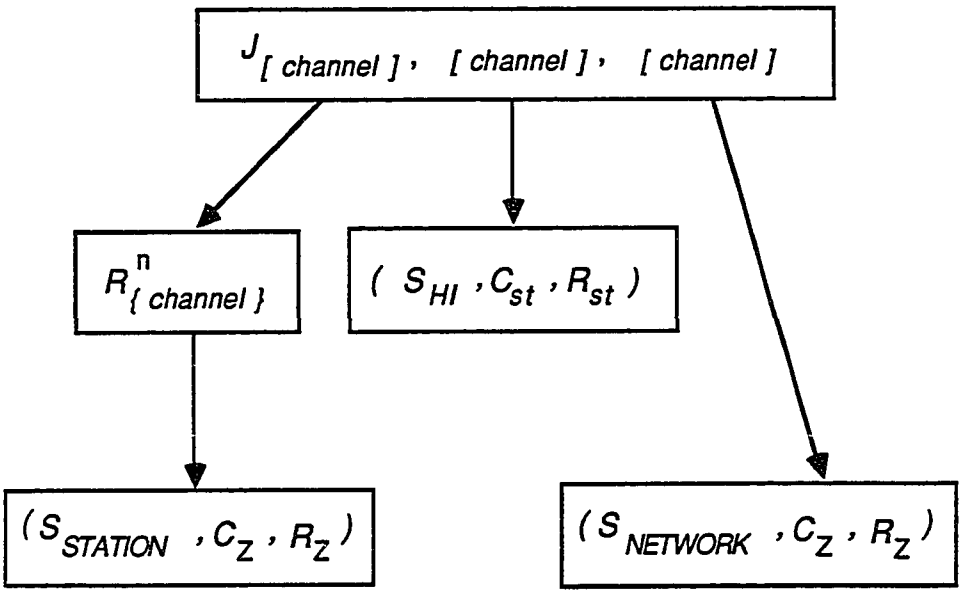


Figure 6.15: Normal/High Priority Network

Number of Stations	Detailed Base Model		Reduced Base Model	
	am-states	*m-states*	*Scenario 1*	*Scenario 2*
2	135	57	30	57
3	841	261	61	147
4	4277	1041	102	276
5	17820	3873	153	444
6	80000	13883	214	651
7	*[a]	*[a]	285	897
8	*[a]	*[a]	366	1182
9	*[a]	*[a]	457	1506
10	*[a]	*[a]	558	1869
11	*[a]	*[a]	669	2271
12	*[a]	*[a]	790	2712

[a] state-space too large to be computed

Table 6.13: State-Space Sizes Obtained Using Various Construction Techniques

Evaluation Results

State-level representations for each scenario were constructed using an extension to METASAN (see the Appendix or [81]) that permits the construction and solution of reduced base model representations. Four state space representations were contrasted: the am-state space, the m-state space, and the reduced base model state-space size for each scenario.

The results are presented in Table 6.13. One can see that for each network configuration considered, the reduced base model construction technique generated significantly fewer states compared to the marking and activity-marking states. Moreover, the rate of growth of the state-space experienced by increasing the number of stations in the network was much smaller for the reduced base model construction technique. In fact, the number of states generated using this technique remained quite small even when the generation of state-spaces using standard techniques became intractable.

The reduced base model representations were then used to investigate the performance of a ten station network. Specifically, expected queue lengths and the fractions of time

Fraction of Full Load	Expected Queue Length	Prob. Blocking	Fraction of Time			
			Idle	*Unprop.*	*Prop.*	*Collision*
.10	.0022	.0000	.8957	.0050	.0992	.0002
.20	.0072	.0001	.7939	.0098	.1955	.0009
.30	.0153	.0005	.6963	.0143	.2867	.0026
.40	.0271	.0013	.6046	.0186	.3711	.0057
.50	.0427	.0027	.5202	.0224	.4472	.0103
.60	.0622	.0049	.4438	.0257	.5138	.0166
.70	.0857	.0081	.3762	.0285	.5705	.0248
.80	.1128	.0125	.3174	.0309	.6172	.0346
.90	.1432	.0182	.2670	.0327	.6543	.0459

Table 6.14: Performance of 10 Station Homogeneous Network

each type of packet was on the bus in steady-state were computed the network for various network loads. The results of these experiments are presented in Table 6.14. In this table, the second column is the expected queue length at each station and the third column is the probability that an incomming message is blocked due to a full queue. The remaining columns give the fraction of time the bus is idle, transmitting an unpropagated message, transmitting a propagated message, and clearing a collision. As can be seen by Figure 6.16, the expected queue length grows with increasing load, but remains fairly short even under heavy load conditions.

The second set of experiments investigates the effect of including a high-priority station in the network. The high-priority station is identical to the normal priority station described earlier, except that the mean time between access tries (i.e. one over the rate of activity *access*) is one-tenth that of a normal priority stations. As before, a ten station network is considered, but now nine stations are of the normal priority category and one is a high-priority type. The results of these experiments are given in Table 6.15. As can be seen by this table and Figure 6.17, the expected queue length of the high-priority station remains very short even when the network is loaded very heavily. Furthermore, while the other stations in the network experienced some increase in their average queue

Fraction of Full Load	Expected Queue Length Normal Prio. Station	Expected Queue Length High Prio. Station
.10	.0022	.0011
.20	.0072	.0043
.30	.0155	.0095
.40	.0275	.0165
.50	.0435	.0251
.60	.0637	.0350
.70	.0878	.0459
.80	.1160	.0575
.90	.1477	.0694

Table 6.15: Performance of a 10 Station Network with 2 Station Classes

length (compared to the homogeneous network considered earlier), this increase was slight compared to the decrease in expected queue length at the high-priority station.

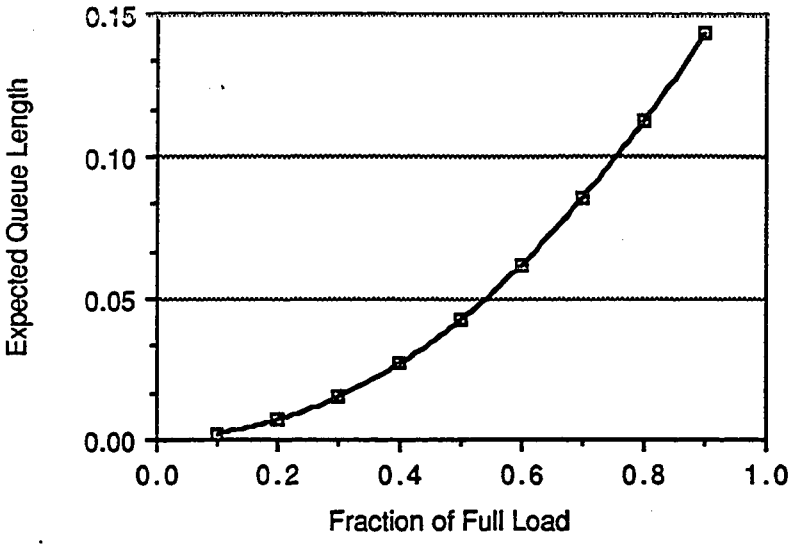


Figure 6.16: Expected Queue Length vs. Load for a 10 Station Homogeneous Network

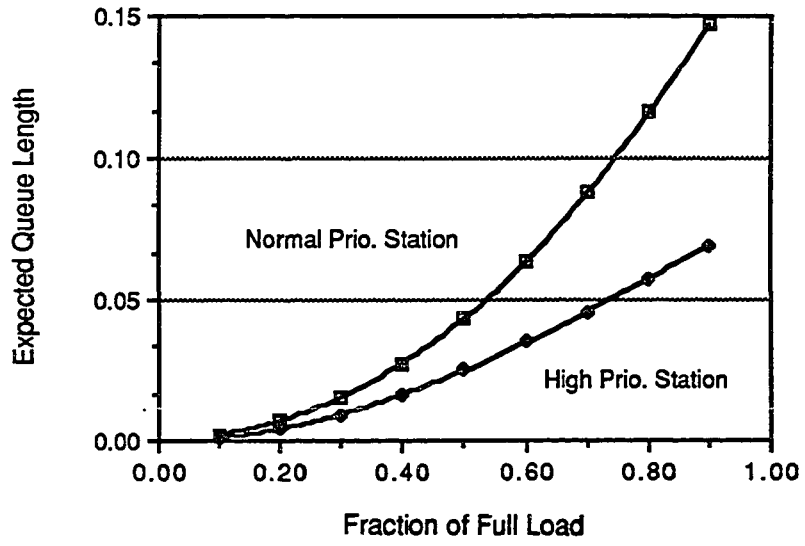


Figure 6.17: Expected Queue Length per vs. Load for a 10 Station Network with 2 Station Classes

CHAPTER VII

CONCLUSIONS AND FURTHER RESEARCH

Contributions

The objective of this research has been to develop construction and solution methods for stochastic activity networks that permit the performability evaluation of realistic distributed systems. To accomplish these objectives, we have:

1. Provided formal definitions of activity networks, stochastic activity networks, and related concepts and used these ideas to investigate when SANs can be used for performability evaluation.
2. Defined a general framework for specifying performance variables at the SAN level.
3. Developed construction techniques that make use of these variables and network characteristics to reduce the size of the base model representation.
4. Investigated the use of traditional stochastic process solution methods, defined a simulation procedure, and developed a reward model solution technique that can be used to solve for the variables defined in Chapter 3.
5. Illustrated the use to these methods by evaluating two distributed systems.

More specifically, with regard to the first objective, we have provided formal definitions for SANs and investigated when they are *well specified.* This condition specifies when the behavior of a stochastic activity network is probabilistically completely specified, and hence, can be used for performability evaluation. With regard to the second objective, we have defined a framework that allows performance variables to be specified at the network level. A detailed discussion of one category of variables within this framework is given,

along with examples of how variables within this category can be specialized to traditional performance and reliability variables.

With regard to the third objective, we have developed construction procedures for detailed and reduced base models, and defined conditions under which these models support a chosen variable and are solvable. Detailed base models support large classes of variables and are useful when the SAN in question is not too complicated (i.e., its base model representation is not too large). Reduced base models are obtainable for certain classes of SANs and variables. When they can be used, they typically have far fewer states then the corresponding detailed base model.

Solution methods were also addressed, as per the fourth objective. In particular, we investigated the use of, and implemented in METASAN (see the Appendix), several solution methods. Steady-state and transient Markov model solution methods are used to obtain instant-of-time variables and Markov reward model solution methods are used to obtain interval-of-time variables. In addition, a solution method to find the expected value of many interval-of-time variables was proposed and implemented in METASAN. Finally, a simulation procedure was developed to obtain solutions for variables when analytic methods can not be applied. Details of the implementation of these solution techniques, as well as some of the construction techniques discussed in Chapter 4, are given in the Appendix.

Finally, as per the fifth objective, the utility of these construction and solution methods was illustrated via two evaluation studies. The first was an evaluation of an industrial network employing the IEEE 802.4 token bus protocol. The study showed that noise bursts and random token losses have minimal impact on network performance at moderate (50%) network loading and manageable impact at very high (85%) loading. The second study was an evaluation of a CSMA/CD local area network, and illustrated the state-space savings that can be achieved through the use of reduced base model construction methods.

Directions for Further Research

Although the research objectives described in the first chapter were achieved, there are still several interesting related research areas that deserve future attention. Broadly speaking,

they can be grouped into areas that further the model construction and solution work presented here and into areas that use SANs for broader purposes.

Regarding model construction and solution, one area of further research is the definition and investigation of other categories of performance variables for use with stochastic activity networks. While a large number of variables can be generated within the activity-marking category presented in Chapter 3, this category does not subsume all conceivable performance variables. In particular, while the value of an activity-marking interval or time-averaged interval variable can be highly dependent on past history, the values assigned to activities and sets of places in the reward structure cannot depend on the past history of the execution of a stochastic activity network. This precludes assigning a single reward to a sequence of activity completions and intervening markings, even though this sequence may represent a single operation in the modeled system.

Similar to the path variables discussed in the Appendix, this new category of variables would assign rewards to completions of sequences of activities and intervening markings and would assign reward-rates to being within particular sequences. Construction procedures could then be developed that generate a process tailored to a particular variable within the new category. This could be done by selectively keeping just enough past history in a state such that sequences could be detected. Conceivably, the replicate and join operations could also be exploited in order to achieve further state-space space savings.

Another area of interest is to apply the reduced base model construction ideas presented in Chapter 4 to the construction of simulation programs from stochastic activity networks. In particular, recall that the simulation procedure described in Chapter 5 treated each activity as an "event type" and, hence, the future event list contained the potential completion times and names of all activities that were active in a current marking. We have observed that, for big models (on the order of a thousand activities, places, and gates), a large percentage (over 70% in some cases) of the total simulation run time is spent checking to see which activities are enabled in a current marking. Since activities within replicated submodels are equivalent with respect to both their enabling conditions and associated rewards, they could be considered as a single event type in the simulation, greatly reducing the amount of time that is spent updating the future event list.

Although the reduced base model construction methods developed in this dissertation have greatly extended the size of models that can be solved analytically (relative to methods that use detailed base models), there is still work to be done to avoid the "state-space explosion" problem. One possible avenue of research would be to investigate the application of reversible stochastic process theory to SANs in a manner similar to that done for product-form queueing networks (e.g., [6,41]). Some attempts in this direction have been made [54] for GSPNs, but have been limited to a particular multiprocessor structure.

It would also be interesting to investigate the use of stochastic activity networks for verification. Verification (in the sense intended here) is a validation method which uses formal proof methods to establish that a system is "correct" with respect to a specification. This type of validation is typically used to establish that certain logical properties always hold for a given realization. While much work has be done concerning the use of traditional Petri nets for verification [78], the generalization of these techniques to extended Petri nets (such as GSPNs, ESPNs, and SANs) is not straightforward. One area of interest would be to identify sub-classes of SANs to which classical techniques could be applied; another area would be to develop new verification methods that could be applied to larger classes of SANs.

It would also be interesting to investigate the use of stochastic activity networks in the design process. While current methods and tools yield results that can be used to make design decisions, the incorporation of these decisions into a next-generation model of a system must currently be done in an *ad-hoc* manner. By investigating the use of model-based evaluation in the design process, one may be able to develop customized tools that permit design, evaluation, and verification to be interleaved in a systematic manner.

Finally, the methods and techniques reported on in this dissertation could be used to evaluate the performability of other distributed systems. While studies to date have been in the areas of computer architecture, computer networks, and factory scheduling, applications may exist in such diverse areas as economics and business planning, sociology, and biology.

APPENDIX

Implementation of Methods via METASAN

Introduction

Both the model construction and model solution methods discussed in this dissertation provide a conceptual method to evaluate a large class of complex, distributed systems. In order to be used in practice, however, they must be embodied in a computer automated tool that facilitates entry of both models and performance variables and automates model construction and solution. METASAN[1], a software package developed at the Industrial Technology Institute, is one such tool. It contains routines for both model construction and solution, and includes both simulation and analysis as solution options.

Relative to the objectives of this dissertation, METASAN has served a dual purpose. First, it provided an environment in which to test the usefulness of basic model definitions, construction procedures, and solution procedures. Second, it was a tool that could be used to evaluate the performability of realistic systems. The two applications presented in Chapter 5 are examples of this use. Other applications include the evaluation of a self-exercising self-checking memory [61] and a BIBD (Balanced Incomplete Block Design) computer network [1].

The development of METASAN paralleled the work on this dissertation. In some cases, theory and algorithms from the dissertation were implemented in METASAN, in

[1] METASAN is a Trademark of the Industrial Technology Institute.

other cases, an initial implementation in METASAN motivated a refinement in an idea that was presented in the dissertation but not yet implemented in METASAN. Thus the performance variables, construction algorithms, and solution algorithms present in METASAN differ slightly from those presented in the previous chapters. This is natural since the purpose of METASAN, relative to the dissertation work, was to provide a test-bed for ideas as well as an efficient performability evaluation tool.

This appendix gives an overview of the current release (version 1.3) of METASAN. More detailed information can be found in the METASAN user's documentation [16]. The remainder of the appendix is organized as follows. The next section describes the high level organization of METASAN. The following section discusses the model construction procedures implemented in METASAN. Likewise, the fourth section describes model solution algorithms implemented. Use of the package is illustrated in final section via its application to a specific evaluation problem.

METASAN Organization

METASAN was designed in a modular manner to allow for the addition of new solution methods as they become available. It is written using UNIX[2] tools (C, Yacc, Lex, and Csh) and currently contains some 37,000 lines of source code. Versions are available for both a SUN-3[3] running SUN UNIX and a VAX[4] running UNIX 4.3 BSD. Steady-state and transient evaluation, via both analysis and simulation, are supported by the tool.

At the highest level, the analyst interacts with METASAN through a menu structure. This menu (Figure A.1) permits access to the two basic files that make up a METASAN model: a structure file and an experiment file. The menu also permits access to the compilers for each of the description files and the solution modules. The structure file is a direct translation of the SAN into a textual form that can be accepted by METASAN. Specification of the desired performance variables and solution algorithm is done via either a simulation experiment file or analytic experiment file, depending on the type of solution

[2] UNIX is a registered trademark of AT&T.

[3] SUN-3 is a Trademark of Sun Microsystems, Inc.

[4] VAX is a Trademark of the Digital Equipment Corporation.

```
#### Copyright 1988 Industrial Technology Institute.  All Rights Reserved. ####
#                                    #                                       #
#  METASAN (Version 1, Release 3)    #  METASAN Simulator Selections         #
#                                    #                                       #
#  F)  Set Files                     #  no trace options set                 #
#  O)  Set Options                   #  no verbosity set                     #
#  S)  System Commands               #  no checkpoint set                    #
#                                    #  fast execution                       #
#  s)  compile structure description #                                       #
#  e)  compile experiment description#                                       #
#  c)  convert SAN to SAS            #                                       #
#  r)  bind and solve model          #                                       #
#  u)  update files and solve        #                                       #
#                                    #                                       #
#  vs) edit structure file           #  path: /users/bs/msan                 #
#  ve) edit experiment file          #  model directory: multi               #
#  vi) edit a file                   #  structure file:  multi1              #
#                                    #  experiment file: structure           #
#  q)  quit METASAN                  #  object file:     demo                #
#####################################################################################

   Command:
```

Figure A.1: Main METASAN Menu

method used. Each of these files will be described in more detail in a following section. Model construction consists of describing the structure of the system to be modeled using the editor (option vs), compiling the description (option s), describing the experiment file (option ve) and compiling the experiment file (option e). The result of these actions is a machine understandable description (a collection of C data structures and procedures) of the system to be modeled and the desired performance variables. This representation can then be directly executed, if the solution method is simulation, or be used to generate a state-level representation (option c), if the model is to be solved via analysis. Selecting model solution (option r) binds the machine readable description to the correct solution routines and then executes them to obtain the selected performance variables. This process can be automated by use of the "update files and solve" option (u). This option selectively executes the commands described above, as necessary, without user intervention. The decision as to whether a command needs to be executed is made by looking at the last change date of the various model files. This makes it very easy to make changes to a single model file, recompiling only those files necessary to insure the integrity of the solution.

A flowchart of this process is given in Figure A.2. Output from the solver depends on the solver chosen, but contains either estimates or exact values of the chosen performance variables.

Items in the pictured menu that are identified by a capital letter designate lower level menus. "Set Files" allows the user to specify each of the files associated with a particular model. "Set Options" offers a variety of options which, depending on the solution module selected, allow the user to set various trace and debug options, run the model in the background or on remote machines, and direct the output to several locations. "System Commands" consists of a collection of commands to aid the user in creating new model directories, removing or copying description files, etc.

Model Construction

In the context of METASAN, model construction consists of describing both the stochastic activity network and chosen performance variables in a manner understandable to the package and translating these descriptions into a form that is expected by the solution modules. This is accomplished in METASAN via three input languages and their corresponding compilers.

Sanscript Model Description File

The SAN description language, *Sanscript*[5], allows the analyst to specify the SAN in a textual form understandable to the SAN compiler. *Sanscript* permits easy specification of complex enabling predicates, activity time functions, reactivations functions and gate functions. The language is illustrated via its representation of the SAN of Figure A.3.

Figure A.4 is the *Sanscript* representation of the portion of this stochastic activity network that represents variations in the internal state and environment of the system; activities representing fault occurrences are not represented. At a high-level, a *Sanscript* description consists of four parts: a header, local variable declarations, definition of all the

[5] *Sanscript* is a Trademark of the Industrial Technology Institute.

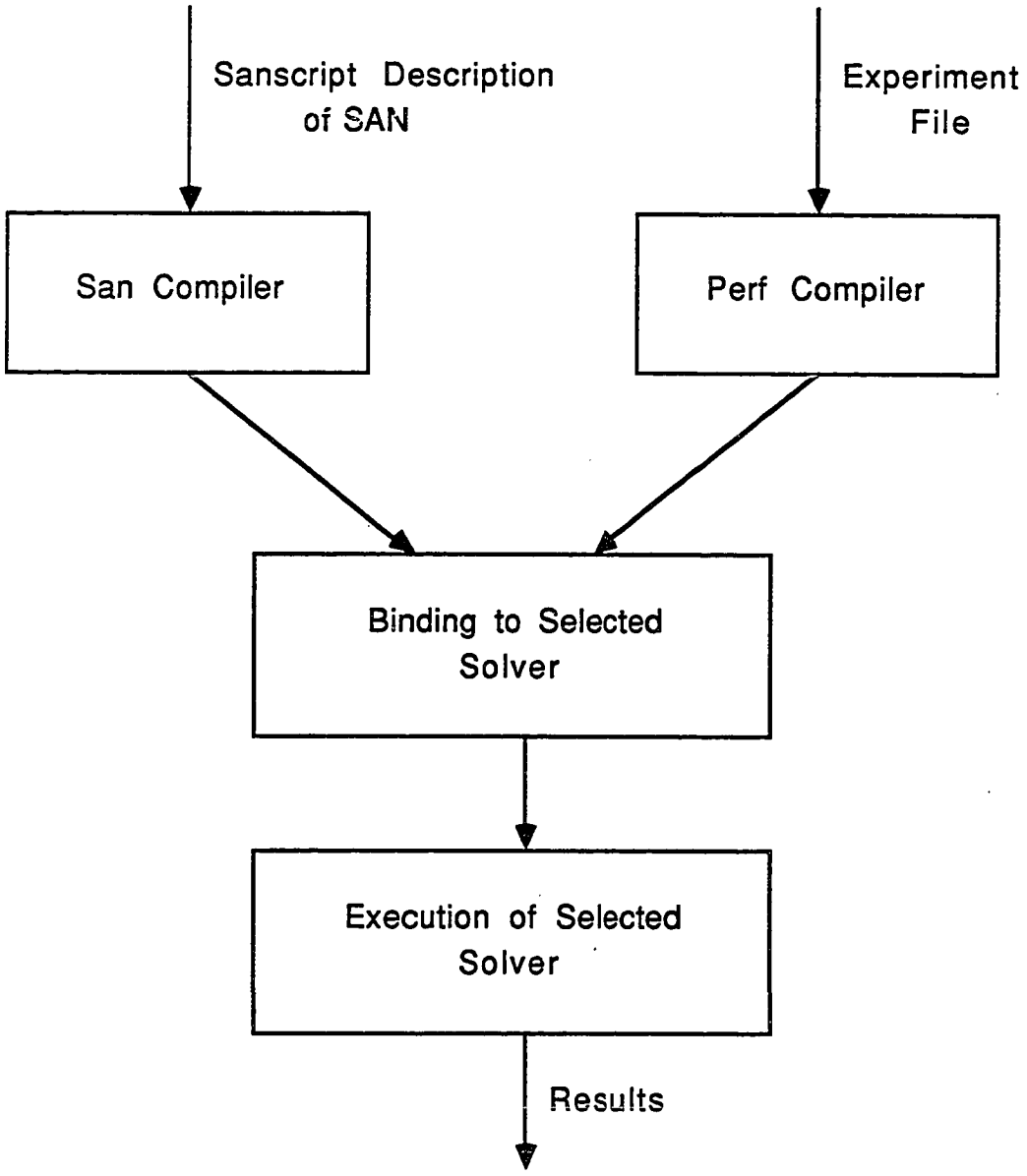


Figure A.2: METASAN Flow Diagram

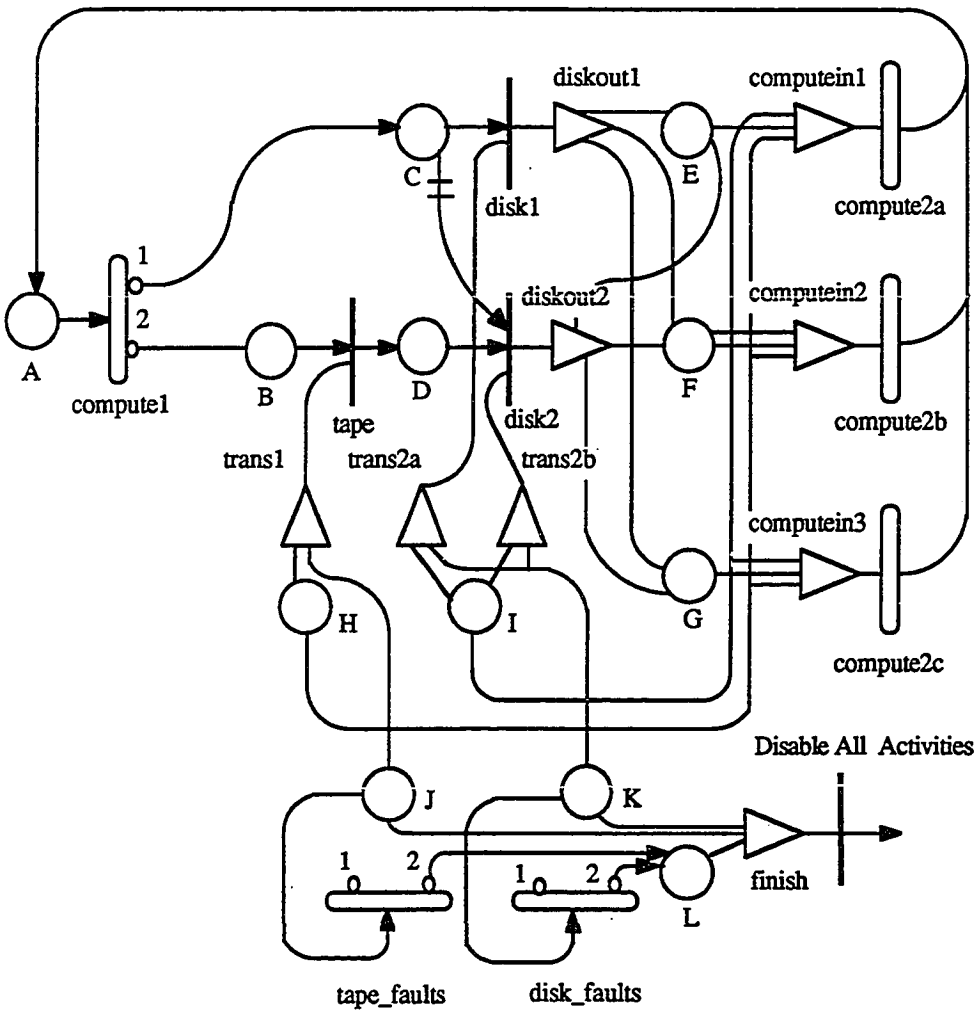


Figure A.3: Example Stochastic Activity Network

```
DESCRIPTION multi1;

OBJECTS

place:          A,7; B,0; C,0; D,0;     /* definition of places */
                E,0; F,0; G,0; H,0;
                I,0; J,2; K,3;
activity:       compute1;               /* definition of activities */
                tape;
                disk1; disk2;
                compute2a; compute2b;
                compute2c;
input_gate:     computein1;             /* definition of input gates */
                computein2;
                computein3;
                Trans1;
                Trans2a;
                Trans2b;
output_gate:    diskout1;               /* definition of output gates */
                diskout2;

SPECIFICATION

compute1        [ cases 1: prob { .3 } C;
                        2: prob { .7 } B;
                  input A;
                  exp { MARK(A) };
                  react { 1 } { 1 }; ]

tape            [ case D;
                  inputs B; Trans1;
                  inst; ]

disk1           [ case diskout1;
                  inputs C; Trans2a;
                  inst; ]

disk2           [ case diskout2;
                  inputs ~C; D; Trans2b;
                  inst; ]

compute2a       [ case A;
                  input computein1;
                  determ { MARK(E) };
                  react { 0 } { 0 }; ]

compute2b       [ case A;
                  input computein2;
                  determ { MARK(F) };
                  react { 0 } { 0 }; ]
```

Figure A.4: Example Description File

```
compute2c       [ case A;
                  input computein3;
                  determ { MARK(G) };
                  react { 0 } { 0 }; ]

diskout1        [ outputs 1: E; 2: F; 3: G;
                  func { if (X1 == 0) X1 = 1;
                         else if (X2 == 0) X2 = 1;
                         else if (X3 == 0) X3 = 1; } ]

diskout2        [ outputs 1: E; 2: F; 3: G;
                  func { if (X1 == 0) X1 = 2;
                         else if (X2 == 0) X2 = 2;
                         else if (X3 == 0) X3 = 2; } ]

computein1      [ inputs 1: I; 2: E; 3: H;
                  pred { X2 >= 1 }
                  func { if (X2 == 2) { X1 = X1 - 1; X3 = X3 - 1; }
                         else X1 = X1 - 1;
                         X2 = 0; } ]

computein2      [ inputs 1: I; 2: F; 3: H;
                  pred { X2 >= 1 }
                  func { if (X2 == 2) { X1 = X1 - 1; X3 = X3 - 1; }
                         else X1 = X1 - 1;
                         X2 = 0; } ]

computein3      [ inputs 1: I; 2: G; 3: H;
                  pred { X2 >= 1 }
                  func { if (X2 == 2) { X1 = X1 - 1; X3 = X3 - 1; }
                         else X1 = X1 - 1;
                         X2 = 0; } ]

Trans1          [ inputs 1: H; 2: J;
                  pred { X2 > X1 }
                  func { X1 = X1 + 1; } ]

Trans2a         [ inputs 1: I; 2: K;
                  pred { X2 > X1 }
                  func { X1 = X1 + 1; } ]

Trans2b         [ inputs 1: I; 2: K;
                  pred { X2 > X1 }
                  func { X1 = X1 + 1; } ]
end.
```

Figure A.4: Example Description File, Cont.

primitives used, and a specification of all functions, values, and interconnections associated with each primitive.

In our example, the header specifies that the name associated with this SAN is `multi1`. The local variable declaration section is not used in this example, but can be used to declare variables that are local to specific gate, case and reactivation functions. Next comes the declaration of primitives. Note that the initial marking for each place is specified directly after the place name. The specification section follows. In any function, the current marking of the place can be referenced by using the notation `MARK(`*place*`)`, where *place* is the name of the desired place. For an example specification, refer to the definition of *compute*1. As can be seen in the graphical representation, timed activity *compute*1 has two cases, one connected to place C, and one to place B, and a single input place A. The activity time distribution is specified to be exponential with parameter `MARK(`A`)` and the case distribution is specified such that there is a 30% chance of choosing case 1 and 70% chance of choosing case 2 upon completion of the activity. The reactivation function is specified so that for every marking (denoted by the 1 in the first set of brackets after `react`) the set of reactivation markings is all markings (denoted by the second 1 in brackets). In other words, the activity is reactivated at every state change. The reason for this will be discussed in the example section. Note that any expression that evaluates to the correct type (real for prob and exp, boolean for react) may be given between the curly brackets; complex interactions, activity time descriptions, reactivation functions, and case distributions can be represented easily via a few C statements.

A wide range of activity time distribution types are available, representing all service distributions normally used in evaluation. An activity time distribution function of `inst` is used to denote an instantaneous activity. Of course, the choice of distribution affects the nature of the underlying stochastic process and the solution methods that may be used. Complex reactivation functions can be represented by specifying several pairs of predicates, the interpretation being that the activity is reactivated if, a) the activity was activated in the subset of the reachable markings specified by the first predicate, and b) a marking in the subset specified by the corresponding second predicate is reached. Again, if the activity is enabled in this second marking it is immediately restarted.

Gates are specified in a similar manner. For example, see the description of the input gate *computein*1 in both the graphical and textual representation. The input places associated with the gate are specified as an indexed list. This abbreviated notation allows place names to be abbreviated in the predicate and function descriptions according to the following rule: `Xi` denotes the `i`-th place specified in the associated input or output (in output gates) place specification. An example of this can be seen in both the predicate and input function description of *computein*1. Note again that any legal C statements may be placed within the brackets, as well as the abbreviated notation described above and the *MARK* function notation. Output gates are specified in an identical manner except that the keyword `outputs` is used in place of `inputs` and there is no associated enabling predicate.

A macro-preprocessor is automatically invoked on execution of the compiler, and can be used to define subnetworks in a parameterized manner. Once this is done, specific subnetworks can be constructed by a single macro call. This feature is particularly useful when applied to stochastic activity networks which contain numerous similar subnetworks (e.g., nodes in a computer network) that are replicated many times. Construction of these subnetworks directly in *Sanscript* would be tedious.

After the specification of the SAN in *Sanscript* is complete, it is passed to the SAN compiler to be translated into an internal form understandable by the solution modules. The SAN compiler is written in Yacc (yet another compiler compiler), Lex and C.

Simulation Experiment File

The SAN description language is independent of the chosen solution method. In contrast, there are two languages for the specification of performance variables to the package. The first language is used when the intended solution method is simulation (either terminating or steady-state); the second is used to specify variables when an analytic solution method will be used.

Since simulation puts few limitations of the types of variables that may be estimated, the class of variables that may be used is much larger than the activity-marking oriented variables that were described in Chapter 3. Before discussing the syntax and options of

the simulation experiment file, it helps to describe the allowable variables in more detail. In particular, we define a class of variables that can capture a "behavior" associated with a sequence of activity completions and reached markings. An example of a behavior of this type is a recovery operation in a multiprocessor, which may be modeled as a subnetwork with several places and activities. Here, one may wish to determine the amount of time spent doing recovery operations during a bounded or unbounded interval, or the number of times a particular sequence of activities completes during the interval. Since these behaviors are represented by a particular sequence of events in the subnetwork, they cannot be detected by assigning rewards to single activity completions and/or markings of sets of places.

The "path variables" presented here capture behaviors that are associated with sequences of activities. Recall (from Chapter 2) that a *path* is a sequence of configurations $C_1, C_2, C_3, \ldots, C_n$ such that for each pair of configurations $< \mu_i, a_i, c_i >< \mu_{i+1}, a_{i+1}, c_{i+1} >$ the completion of a_i and choice of c_i in μ_i results in μ_{i+1}. A path may be traversed during an execution of a SAN. The following discussion makes this more precise. A *completion of a configuration* $< \mu, a, c >$ occurs at time $t + s$ when during an execution of a SAN the configuration is found in a step $(< \mu, a, c >, t, s)$ in some set of steps. An *initiation of a path* occurs when the SAN reaches the marking associated with the first configuration in the path. A *completion of a path* occurs when the SAN completes the final configuration in the path after completing each configuration in the path in the order specified without reaching any intermediate configurations other than those specified in the path. A *traversal of a path* is the act of first initiating the path and then completing the path.

These definitions allow us to introduce a set of performance variables to study the behavior of the SAN. Note that they are indexed either by time or number of occurrence so that, if they converge in distribution, their limit can be studied.

$N_T(f,t)$ a measure of the value of the function f at time $t+$. f can be any real-valued function on the marking of the net. Examples include the marking of a single place and the sum of the markings of several places.

$T_b(S,n)$ the time between $n-1th$ and nth completion of any path in a set of paths S.

$T_w(S,n)$ the time to traverse the *nth* path to complete in a set of paths S.

$I(T,t)$ an indicator random variable denoting the event of being in any configuration in a set of configurations T at time $t+$.

$N_{TR}(S,t_1,t_2)$ the sum of the number of traversals of any path in a set of paths S during $[t_1,t_1+t_2]$.

$T_I(S,t_1,t_2)$ the sum of the time spent traversing all paths in a set of paths S during $[t_1,t_1+t2]$.

Note that there is a chance, depending on the definition of the sets of paths, that two paths may complete at the same time. Since some of the random variables defined above are indexed by the order of completion of the paths, a rule must be given to resolve any ambiguity as to which path is which. To do this we use the convention that if two paths complete at the same time, the path which took the longest time to traverse will be counted as completing first.

By varying the index of any of the first four variables above, one can construct a sequence of random variables. Under certain conditions, the distributions of the random variables in this sequence may converge to a single (steady-state) distribution. More precisely, following the terminology in [49], for a sequence of random variables $\{V_n, n = 1,2,3,\ldots\}$, we define

$$F_V(x) = \lim_{n\to\infty} Pr\{V_n \leq x\}.$$

In terms of the random variables defined above, we have:

$F_{N_T(f)}(x) = \lim_{t\to\infty} Pr\{N_T(f,t) \to x\}$ the distribution of the value of function f in steady-state.

$F_{T_b(S)}(x) = \lim_{n\to\infty} Pr\{T_b(S,n) \to x\}$ the distribution of the time between completions of any path in a set of paths S in steady-state.

$F_{T_w(S)}(x) = \lim_{n\to\infty} Pr\{T_w(S,n) \to x\}$ the distribution of the time between initiations and completions of any path in a set of paths S in steady-state.

$F_{I(T)}(x) = \lim_{t\to\infty} Pr\{I(T,t) \to x\}$ the distribution of the event of being in any configuration in a set of configurations T in steady-state.

Instances of the above path variables, along with their corresponding path sets, are specified in a simulation experiment file. A simulation experiment file specifies the paths sets, variables defined on those paths, definitions of any variables derived from previous variables, and desired characteristics of the defined variables (i.e., mean, variance, percentile, or interval). Mean, variance, interval, and percentile estimators with confidence intervals are available.

For example, consider an experiment file (Figure A.5) for the example of the SAN of Figure A.3. Here the goal is to obtain the mean, variance, an interval, and several percentile estimates of the time between firings of activity *compute*1 in the long run, so the steady-state simulator is used. The experiment file requires definitions of configurations, path sets, performance variables, and estimators for the defined variables. Configurations are specified in a compact notation and path sets are built up from sets of configurations. The meta-character "*" can be used to denote any marking, activity, or case, depending on its position. In the example, `(MARK(A) == 0, (*,*))` specifies all configurations such that the marking of A is 0, any activity completes (denoted by the first *), and any case is chosen (denoted by the second *). "==TIMED" and "==INST" can also be used in the activity specification to denote any timed or instantaneous activity, respectively. Path sets are then constructed from these configuration sets. The syntax "||" can be used to "or" configuration sets together at either the `CONFIGURATION` or `PATHSET` level in the experiment file. Although all of the path sets in our example consist of a single configuration set, longer paths can be specified by naming a sequence of configuration sets separated by commas. The interpretation of this notation (in the context of our earlier theory) is that the set of paths included is all paths constructed by selecting a configuration from each of the configuration sets in order. Two additional operators aid in the definition of path sets. "*( )" denotes that the sequence of configurations within the parenthesis must be repeated 0 or more times. "+( )" denotes that the sequence of configurations within the parenthesis must be repeated 1 or more times.

The user then specifies the desired performance variables in terms of the previously

```
PERF percent;

CONFIGURATIONS
        CYCLE  = ( *, (compute1,*) )
        CYCLEV = ( *, (compute1,*) )
        CYCLE1 = ( *, (compute1,*) )
        CYCLE2 = ( *, (compute1,*) )
        CYCLE3 = ( *, (compute1,*) )
        CYCLE4 = ( *, (compute1,*) )
        CYCLEI = ( *, (compute1,*) )

PATHSETS
        Mean  = [ CYCLE ]
        Variance = [ CYCLEV ]
        Percentile1 = [ CYCLE1 ]
        Percentile2 = [ CYCLE2 ]
        Percentile3 = [ CYCLE3 ]
        Percentile4 = [ CYCLE4 ]
        Interval = [ CYCLEI ]

MEASURED VARIABLES
        SS_TB ( Est_Mean, Mean, 100, 3200);
        SS_TB ( Est_Var, Variance, 100, 3200);
        SS_TB ( Per_05, Percentile1, 100, 3200);
        SS_TB ( Per_35, Percentile2, 100, 3200);
        SS_TB ( Per_70, Percentile3, 100, 3200);
        SS_TB ( Per_100, Percentile4, 100, 3200);
        SS_TB ( Interval_1, Interval, 100, 3200);

ESTIMATIONS
        MEAN( Est_Mean, .01, .95)
        VARIANCE( Est_Var, .01, .95)
        PERCENTILE( Per_05, .03, .95, .05)
        PERCENTILE( Per_35, .01, .95, .35)
        PERCENTILE( Per_70, .01, .95, .70)
        PERCENTILE( Per_100, .01, .95, 1.0)
        PINTERVAL( Interval_1, .01, .95, .5, 1.5 )

HALTING CONDITIONS

RESOLUTION RULES
       { compute_2a, compute_2b, compute_2a }

TYPE
       steady_state
end.
```

Figure A.5: Example Simulation Experiment File

defined path sets. Each of these variable definitions is itself implemented in a low-level language that makes it easy to create new variables. Experiment file section ESTIMATIONS allows for specification of the estimator to be defined on the variables. Currently, mean, variance, interval, and percentile estimators are supported, all with confidence interval estimation. Examples of the syntax for each of these estimators can be found in Figure A.5. In each case the user specifies first the relative confidence interval width (.01 in the example) and confidence level (95%). For the interval estimator the user also specifies the bounds for the interval to be considered. In the example, the bounds indicate that the probability that the time is between .5 and 1.5 is to be estimated. For the percentile estimator the user specifies the percentile for which the estimate is desired (.05, .35, .70, and 1.0 in the example).

An identical experiment file syntax is used in conjunction with the terminating simulation solver. The simulation experiment file is then passed to the simulation perf compiler, where the specification is translated into a second collection of C data structures and procedures that are understandable to a simulation solver. Experiment files used in conjunction with analytic solvers are now discussed.

Analytic Experiment File

Analytic variables are specificed via a second experiment file language. Although the language does not refer to them explicitly, it can be used to specify the activity-marking oriented variables discussed in Chapter 3. In the language, variables to be solved for are specified by writing them in terms of possible "solution vectors" for a given solution method. For example, a solution vector produced by the steady-state state occupancy solver is the probability of being in a state (or, for ergodic systems, the fraction of time spent in that state) in the long run. Solution vectors for the transient solver are the state occupancy probabilities at specific times. Variables that are expressed directly in terms of these solution vector are called *basic variables.* Basic variables can then be used, together with constants, to construct *derived variables.*

The analytic experiment file is broken up into six parts: a header, solution type specification, state type specification, input section, output section, and report section. An

example experiment file is given in Figure A.6. The header section specifies that the name of experiment file is `structure`. The solution type specification specifies the solution method to be used. Five solution methods are available: `direct_steady_state`, `iterative_steady_state`, `transient`, `expected`, and `reward_rate`. Details of these solvers are discussed in the section entitled "Model Solution". The state type specification specifies the type of base model that is to be used. An entry of `m_state` specifies that the m-behavior is to be used; an entry of `am_state` specifies that the am-behavior is to be used. Reduced base models are not supported in Version 1.3 of METASAN.

Next comes the input section. The input section is used to pass parameters to the chosen solver, and hence, varies according to the solver used. No parameters are needed for the `direct_steady_state` solver. The same parameter is used for the `iterative_steady_state`, `transient`, and `expected` solvers, and is a specification of the desired relative precision of the results. This is specified using the syntax "acc=*value*" where *value* is the relative precision of the solution vector. For example, if *value* = .001 then the answers are guaranteed to two decimal places. A value of .00001 guarantees four decimal places of accuracy. As the level of accuracy is increased, the running time of the solver increases. The `reward_rate` solver requires the specification of a reward rate for each base model state.

The output section contains the specification of the variables to be computed. The first subsection specifies the basic variables in terms of solution vectors. The solution vector used in the example experiment file is `e_time`, which is the expected time that the model is in possible states during a specified period. For example, the assignment "`time_10[1][2] = (e_time(10), {MARK(J)==1 && MARK(K)==2});`" specifies that the variable "`time_10[1][2]`" is to be set to the expected amount of time that the marking of place J is 1 and the marking of place K is 2 during the interval $[0, 10]$. As can be seen from the example, loops can be used to make the assignment of multiple variables easy. The "`DERIVE`" subsection permits the computation of derived variables from the basic variables computed in the "`ASSIGN`" section. In the example analytic experiment file, the expected reward for various utilization periods is computed by multiplying the expected time in each base model state by the throughput in that state. Finally, the

```
EXPERIMENT structure;
SOLUTION expected;
STATE_TYPE m_state;

INPUT
  acc = .000001;

OUTPUT
  LOCAL
   { int i, j;
     double reward10  = 0;  double reward50  = 0;
     double reward100 = 0;  double reward200 = 0;
     double time_10[3][4];  double time_50[3][4];
     double time_100[3][4]; double time_200[3][4];
     double throughput[3][4]; }

  ASSIGN
    for { i=1; i<=2; i=i+1 } [
       for { j=1; j<=3; j=j+1 } [
          time_10[i][j]  = (e_time(10), { MARK(J) == i && MARK(K) == j });
          time_50[i][j]  = (e_time(50), { MARK(J) == i && MARK(K) == j });
          time_100[i][j] = (e_time(100), { MARK(J) == i && MARK(K) == j });
          time_200[i][j] = (e_time(200), { MARK(J) == i && MARK(K) == j });
       ]
    ]

  DERIVE
    { throughput[2][3] = 1.409;
      throughput[1][3] =  .713;
      throughput[2][2] = 1.177;
      throughput[1][2] =  .710;
      throughput[1][1] =  .588;
      throughput[2][1] =  .589;

      for (i=1; i<=2; i=i+1) {
        for (j=1; j<=3; j=j+1) {
          reward10 += throughput[i][j] * time_10[i][j];
          reward50 += throughput[i][j] * time_50[i][j];
          reward100 += throughput[i][j] * time_100[i][j];
          reward200 += throughput[i][j] * time_200[i][j];
        }
      }
    }

REPORT
  reward10, reward50, reward100, reward200;
end.
```

Figure A.6: Example Analytic Experiment File

"REPORT" section is a list of the variables (basic or derived) to be reported to the user.

Base Model Construction

When an analytic solution method is used, a base model representation must be constructed before the model can be solved. The construction methods implemented in METASAN differ slightly from those presented in Chapter 4. In particular, an intermediate state-level representation is generated from SAN and used to construct the base model. This representation was used in earlier work regarding stochastic activity networks [66], and is referred to as a *stochastic activity system* (SAS). This representation provides a common intermediate representation that can be used to generate activity-marking and marking detailed models, but is not useful for generation of reduced base models.

Stochastic activity systems can be regarded as probabilistic extensions of Keller's "named transition systems" [43]. The most detailed description of an activity system's behavior are its *state-activity* sequences, i.e., for a given state, the possible sequences of alternating states and activities that can result from a finite number of applications of the transition relation. After the SAS corresponding to the SAN model is generated (see [81] for details), a check is made to determine the stochastic nature of the associated base model representations. The nature can be determined directly from the structure of the SAS. If the behavior is Markov, the stochastic activity system representation can be used to generate a base model for analytic solution, if not, simulation should be used.

Model Solution

Given that the stochastic activity system is Markov, a detailed base model representation (marking or activity-marking) can be generated. The choice of base model type is specified in the analytic experiment file, and discussed in a previous section. After the base model is generated, the prescribed solution is carried out. This section describes each available analytic solution method.

Steady-state instant-of-time variables can be formulated in terms of the steady-state state occupancy probabilities of the resulting stochastic process. Two methods for obtaining these steady-state state occupancy probabilities are available in METASAN. The

first is a direct approach based on the LU decomposition technique [28]. The implementation (the `direct_steady_state` solver) uses the improved generalized Markowitz pivoting strategy [75] to help preserve the sparsity of the matrix. An estimate of the relative error of the result due to machine round-off is also generated from the condition number of the matrix [28] and reported to the user. The direct solver can be used with state spaces of up to 10^4 states, depending on the on the sparsity pattern of the matrix. The solution time is on the order of hours for large systems (a 1000 state system takes approximately 10 minutes to be solved on a VAX-11/785).

An iterative approach is available and is typically used when the matrix representation is too large to be solved by the direct method. The method implemented is based on the Gauss-Siedel technique [28]. Its implementation (the `iterative_steady_state` solver) is straightforward and solutions can be obtained at small computational cost. Then main drawback of the technique is that it does not converge for all matrices and initial guesses. While necessary and sufficient conditions exist to determine whether the method will converge for a particular matrix and initial guess [28], they are as computationally complex as determining the solution and, hence, are not implemented. Instead, the can user enable the reporting of intermediate results, if non-convergence is suspected, to empirically access the situation. The iterative solver can be used for models with state spaces up to five times larger than the spaces allowed by the direct solver. Both the direct and iterative methods make use of sparse matrix representations to reduce storage requirements.

Instant-of-time variables for particular times are determined using the randomization technique proposed by Gross and Miller [33,67]. The approach is based on the known technique of subordinating a Markov chain to a Poisson process (see [14] for example). As implemented (in the `transient` solver), randomization provides a computationally efficient solution for the approximate transient state occupancy probabilities by conditioning on the number of state transitions that may occur during the bounded interval under consideration. Details of the method are given in Chapter 5. As with the steady-state solution methods, sparse matrix techniques are used. In addition, estimates of both the error due to truncating the series and the error due to machine round-off are reported to the user. The randomization technique is implemented so that the computation of state

probabilities at several different times for the same model takes about the same processing time as the computation for the largest of the times. The user can benefit from this feature by specifying more than one time in the input section of the experiment description.

Interval-of-time and time-averaged interval-of-time variables are solved for using reward model solution techniques. Two reward model solvers are provided in the package. The first (the `reward_rate` solver) determines the complete distribution of reward for base models that are acyclic. The technique employed is a variation on a technique proposed by Goyal and Tantawi [32]. The implementation first computes a conditional performability distribution for each trajectory type using the technique described in [32]. The distribution of reward is then obtained via knowledge of the probabilities of trajectory types [25].

If only the expected reward is needed, the solution for expected reward described in the third section of Chapter 5 is used (the `expected` solver). The method is applicable to both cyclic and acyclic methods, as long as the reward model has a finite lifetime. Recall, from Chapter 5, that the method requires the determination of one transient solution and two steady-state solutions. The randomization and LU decomposition techniques described earlier are used to do this. As with the other solvers, the estimates of the errors induced due to round-off and truncation are reported.

Simulation is a useful solution method when complex reactivation functions are specified, activity time distributions are general, the desired performance variables are sufficiently complex, or the state space of the underlying stochastic process is extremely large or infinite. To fill this need, METASAN provides facilities for both terminating (transient) and steady-state simulation.

The implementation is similar to that described in Chapter 5, except that completions of instantaneous activities are considered as events. While this makes the simulator less efficient, it allows one to specify path variables that have configurations with instantaneous activities. Two methods for confidence interval estimation are supported. The first is an iterative method based on the replication approach, and is used for terminating simulations. Using this method, one specifies the relative precision and level of confidence desired as part of the experiment file input. Confidence intervals for steady-state simula-

tion are currently determined using an iterative batching procedure, where the user must specify the length of initial transient, batch size, relative precision desired, and level of confidence desired.

Example

Consider a distributed system where all resources needed are available locally except tape and disk drives. Whenever a disk or tape drive is needed, the processor requests one from a common pool. Time to process such requests is negligible, and the requests are either immediately granted, or the process is blocked. As resources become available, they are allocated to blocked processers in a FIFO manner. Each processor is running an identical application program, whose goal is to process blocks of data. Each block requires processing consisting of the following steps (see Figure A.7): 1) computation for an exponentially (with parameter β) distributed amount of CPU time, 2) allocation of either a disk and tape, or just a disk, with fixed probabilities p and 1-p respectively, 3) computation for a deterministic amount of CPU time equal to ($\alpha \times$ the number of resources requested), and 4) release of all resources allocated. The disk and tape drive are used only for temporary storage, and hence, are not specific to any processor. Upon completion of the processing of a block of data, the processor immediately begins processing another block.

Faults can occur due to the failure of a disk or failure of a tape drive. In each case, the fault may be covered (i.e. the system degrades successfully to a less productive structure state) or it may result in a total loss of processing capability (i.e. total system failure). We assume further that faults in a tape drive and a disk occur as a Poisson process with rates λ and γ respectively. The performance variable considered is the number of blocks that are processed during a finite utilization period of t hours.

Construction of SAN model of system

Performability evaluation requires the construction of a SAN that corresponds to the system being modeled and meets the characteristics required by the particular solution

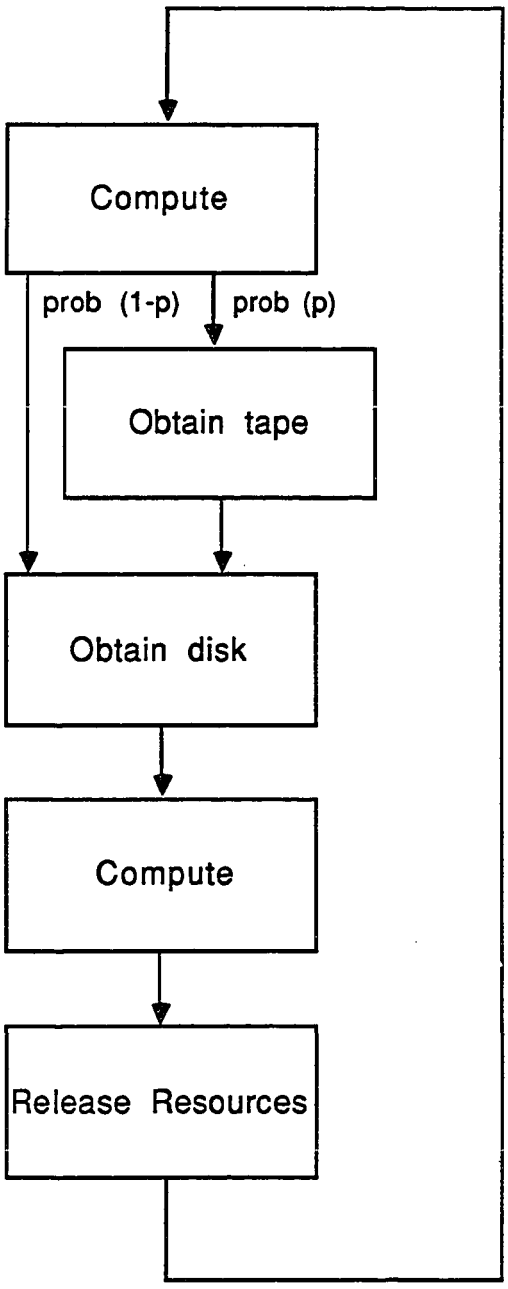


Figure A.7: Processing Algorithm

algorithm. A seven processor case is examined. A SAN that meets these requirements is found in Figure A.3. Here, tokens represent the jobs executing on the processors and resources (tape and disk drives). Tokens in places A,B,C,D,E,F, and G represent the state of jobs executing on the processors. The marking on the diagram is the initial marking and corresponds to the state where all processors are executing the first compute in the sequence of events. Since the length of this compute time is exponentially distributed with parameter β for each processor, all first compute times can be represented via a single activity. The activity time distribution function for *compute*1 is, thus, exponential with parameter $\beta * MARK(A)$. Since the rate at which *compute*1 completes is determined by the number of tokens in place A, it should be reactivated at each state change to insure that the correct rate is always used. Each completion of *compute*1 corresponds to a processor completing the first compute. When this occurs, the process either requests both a tape and disk or a disk only. This choice is represented by the cases associated with *compute*1, where case 1 is chosen with probability (1-p) and represents a request for a disk only; case 2 is chosen with probability p and represents a request for a tape and disk.

These requests are processed by instantaneous activities *disk*1, *disk*2, *tape*, and places H and I. Place H represents the number of allocated tape drives. Place J represents the number of functioning tape drives. Input gate $Trans1$ determines whether there is an available tape drive. Completion of the activity *tape* results in the allocation of a tape drive to the requesting process and the addition of a single token to place H. The number of tokens in place I represents the number of allocated disk drives and the tokens in place K represent functioning disk drives. Input gates $Trans2a$ and $Trans2b$ determine if there is an available disk. Similarly, completion of *disk*1 or *disk*2 represents the allocation of a disk drive to the requesting process. The function of gates *diskout*1 and *diskout*2 is to keep track of the resources that each process possesses during the second compute phase. Two tokens are placed in an output place if the process has both a tape and disk drive; one token signifies the process possesses only a disk. Activities *compute*2*a*, *compute*2*b*, and *compute*2*c* represent the second compute in the algorithm and have deterministically distributed activity times. The compute times cannot be represented by a single activity

since the deterministic distribution is not memoryless. An activity is needed for each process in this phase. Since the maximum number of functioning disk drives in this example is three, only three processes can be in the second compute phase concurrently. Hence, only three activities are needed to represent this phase. Completion of each of these three activities represents the completion of the second compute for a process. The action of the input gate for the activity is to subtract the appropriate tokens from H and I to signify the release of the allocated resources. In addition, a token is added to the output place of the activity (place A) to indicate that the process is again beginning the first compute in the algorithm.

The arrival of faults and (possible) recovery of the system is represented by the remaining places, input gates, and activities. Here places J and K represent the number of fault-free tape and disk drives respectively. Faults arrive to the system upon completion of activities *tapes_faults* and *disk_faults*. The activity time distributions of these activities are exponential with rates $\lambda * MARK(J)$ and $\gamma * MARK(K)$ respectively. Selection of case 1 of either activity represents successful recovery. In this case, one token is subtracted from the appropriate place to indicate the failure of the corresponding resource. Probabilities c_t and c_b are associated with case 1 of activities *tape_faults* and *disk_faults*, respectively. Selection of case 2 represents unsuccessful recovery. If this occurs, a token is placed in L. Completion of the instantaneous activity *Disable* signifies total system failure. This occurs when either recovery from a fault is unsuccessful (signified by a token in L) or when the pool of functioning resources is exhausted (zero tokens in J or K). When either of these events occur, instantaneous activity *Disable* completes and removes all tokens in the network. No blocks are processed in this state.

Model Construction and Solution

In order to solve for the specified performance variable, we decompose the SAN into two submodels, a *performance submodel* and a *structure submodel* as described in Chapter 4. In Figure A.3, the places, activities, and gates above and including J and K comprise the performance submodel. The places, activities, and gates below and including J and K

MARK(J)	MARK(K)	$E[T_b(S)]$	Half Width	$1/E[T_b(S)]$
2	3	0.710	.003	1.409
1	3	1.403	.004	0.713
2	2	0.850	.001	1.177
1	2	1.408	.004	0.710
1	1	1.701	.002	0.588
2	1	1.699	.001	0.589

Table A.1: Results from Performance Submodel

comprise the structure submodel. Places J and K are common places (again, see Chapter 4), and represent the structural configuration of the system.

Evaluation then consists of determining a reward rate corresponding to each structure state, and solving the resulting reward model. In our example, the reward rate is determined by noting that each completion of *compute*1 corresponds to a completion of processing on a block of data. Hence, the rate of completion of block processing is just the inverse of the expected time between completions of *compute*1. In terms of METASAN variables, this corresponds to $1/E[T_b(S)]$, where (the path set) S is defined to be $< *, compute1, * >$. Note that $E[T_b(S)]$ is an estimator defined in the experiment file in Figure A.5. Estimates of this measure were then obtained using the *Sanscript* and experiment files presented together with the steady-state simulation solver. The initial marking of J and K was varied to correspond to each possible structure state. Figure A.8 is the resulting METASAN output for the case $MARK(J) = 2$, $MARK(K) = 1$.

The results for the mean time between completions of *compute*1 for each run are given in Table A.1. "Half Width" refers to the half width of a 95% confidence interval constructed about the estimate. The interpretation of each entry in the last column is the rate at which blocks are processed in that structure state. These rates serve as the reward rates in the subsequent reward model solution. This model was then solved using the expected solver to determine the expected reward derived from operating the system for various utilization periods. These results are given in Figure A.9, and are obtained when $\lambda = .001$, $\gamma = .005$, $c_t = .98$, and $c_d = .99$.

Steady State Simulation Results

Mean Estimations:

Measure Name	Mean	Half Width	Observations
Est_Mean[1.000000] :	1.699547	0.002470	336

Variance Estimations:

Measure Name	Variance	Half Width	Observations
Est_Var[1.000000] :	1.024729	0.007801	336

Percentile Estimations:

Measure Name	Mean	Half Width	Observations
Per_100[1.000000] :	5.256313	0.052502	336
Per_70[1.000000] :	2.166901	0.004645	336
Per_35[1.000000] :	1.212197	0.005511	336
Per_05[1.000000] :	0.202781	0.004589	336

Interval Estimations:

Measure Name	Mean	Half Width	Observations
Interval_1[1.000000] :	0.322656	0.002326	336

Figure A.8: Example METASAN Output

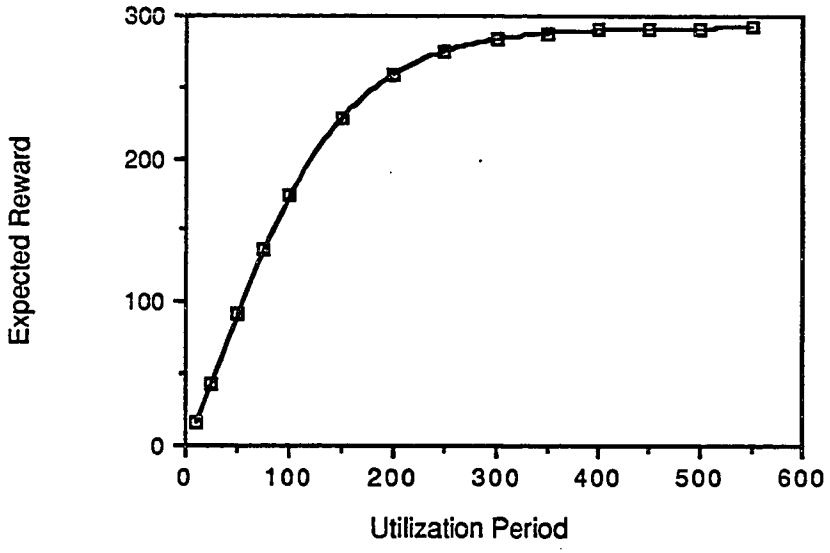


Figure A.9: Expected Reward Obtained for Various Utilization Periods

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] B. E. Aupperle and J. F. Meyer, "Fault-tolerant BIBD networks", in *Proc. 18th International Symp. on Fault-Tolerant Computing*, Tokyo, Japan, June 1988.

[2] A. Avizienis and J. C. Laprie, "Dependable computing: From concepts to design diversity", *Proc. of the IEEE*, vol. 74, no. 5, pp. 629–638, May 1986.

[3] G. Balbo, S. C. Bruell, and S. Ghanta, "Combining queueing network and generalized stochastic Petri net models for the analysis of some software blocking phenomena", *IEEE Trans. on Software Engineering*, vol. SE-12, no. 4, pp. 561–576, April 1986.

[4] G. Balbo, S. C. Bruell, and S. Ghanta, "Combining queueing network and generalized stochastic Petri net models for the analysis of a software blocking phenomenon", in *Proc. International Workshop on Timed Petri Nets*, pp. 208–225, Torino, Italy, July 1985.

[5] G. Balbo, S. C. Bruell, and S. Ghanta, "Modeling priority schemes", in *Proc. 1985 ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pp. 15–26, Austin, TX, August 1985.

[6] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers", *JACM*, vol. 22, no. 2, pp. 248–260, April 1975.

[7] J. P. Behr, N. Dahmen, J. Muller, and H. Rodenbeck, Graphical modeling with FORCASD, in *Proc. Computer Applications in Production and Engineering*, pp. 61–630, Amsterdam: North-Holland, 1983.

[8] B. Beyaert, G. Florin, P. Lonc, and S. Natkin, "Evaluation of computer system dependability using stochastic Petri nets", in *Proc. 11th International Symp. on Fault-Tolerant Computing*, pp. 66–71, Portland, ME, June 1981.

[9] A. Bobbio and A. Cumani, "Discrete state stochastic systems with phase type distributed transition times", in *Proc. International AMSE Conf. on Modelling and Simulation*, pp. 173–192, Athens, June 1984.

[10] V. Chauhan and A. S. Sethi, "Performance studies of token based local area networks", in *Proc. 10th Conference on Local Computer Networks*, pp. 100–107, Minneapolis, MN, October 1985.

[11] J. Y. Chien, "Performance analysis of the 802.4 token bus media access control protocol", in *Proc. Factory Floor Communications Workshop*, pp. 1–39, General Motors Technical Center, Warren, MI, September 1981.

[12] G. Chiola, "A software package for the analysis of generalized stochastic Petri net models", in *Proc. International Workshop on Timed Petri Nets*, pp. 136–143, Torino, Italy, July 1985.

[13] B Ciciani and V. Grassi, "Performability evaluation of fault-tolerant satellite systems", *IEEE Trans. on Communications*, vol. COM-35, no. 4, pp. 403–409, April 1987.

[14] E. Cinlar, *Introduction to Stochastic Processes*, Englewood Cliffs: Prentice-Hall, 1975.

[15] E. Cinlar, *Markov renewal theory*, Advances in Applied Probability, Israel, 1969.

[16] Communications and Distributed Systems Laboratory, *METASAN User's Documentation, Version 1*, Ann Arbor: Industrial Technology Institute, 1987.

[17] A. Cumani, "ESP - A package of the evaluation of stochastic Petri nets with phase-type distributed transition times", in *Proc. International Workshop on Timed Petri Nets*, pp. 144–151, Torino, Italy, July 1985.

[18] L. Donatiello and B. R. Iyer, *Analysis of a composite performance reliability measure for fault-tolerant systems*, Technical Report RC10325, IBM Thomas J. Watson Research Center, Yorktown Hts., NY, January 1984.

[19] L. Donatiello and B. R. Iyer, "Analysis of a composite performance reliability measure for fault-tolerant systems", *JACM*, vol. 34, no. 1, pp. 179–199, January 1987.

[20] L. Donatiello and B. R. Iyer, "Closed-form solution for system availability distribution", *IEEE Trans. on Reliability*, vol. R-36, no. 1, pp. 45–47, April 1987.

[21] J. B. Dugan, *Extended stochastic Petri nets: Applications and analysis*, PhD thesis, Duke University, 1984.

[22] J. B. Dugan, K. S. Trivedi, R. M. Geist, and V. F. Nicola, Extended stochastic Petri nets: Applications and analysis, in *Performance 84*, pp. 507–519, Amsterdam: North-Holland, 1984.

[23] D. Ferrari, *Computer Systems Performance Evaluation*, Englewood Cliffs: Prentice-Hall, 1978.

[24] D. G. Furchtgott, *Performability models and solutions*, PhD thesis, Univ. of Michigan, 1984.

[25] D. G. Furchtgott and J. F. Meyer, "A performability solution method for degradable, nonrepairable systems", *IEEE Trans. on Computers*, vol. C-33, June 1984.

[26] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Englewood Cliffs: Prentice-Hall, 1971.

[27] H. P. Godbersen and B. E. Meyer, "A net simulation language", in *Proc. Summer Computer Simulation Conference*, Seattle, WA, August 1980.

[28] G.H. Golub, *Matrix Computations*, Baltimore: Johns Hopkins University Press, 1983.

[29] A. Goyal and S. S. Lavenberg, "Modeling and analysis of computer system availability", *IBM Journal of Research and Development*, vol. 31, no. 6, pp. 651–664, November 1987.

[30] A. Goyal, S. S. Lavenberg, and K. S. Trivedi, "Probabilistic modeling of computer system availability", *Annals of Operations Research*, vol. 8, pp. 285–306, 1987.

[31] A. Goyal and A. N. Tantawi, *Evaluation of performability for degradable computer systems*, Technical Report RC10529 (Revised), IBM Thomas J. Watson Research Center, Yorktown Hts., NY, December 1984.

[32] A. Goyal and A. N. Tantawi, "Evaluation of performability for degradable computer systems", *IEEE Trans. on Computers*, vol. C-36, no. 6, pp. 738–744, June 1987.

[33] D. Gross and D. R. Miller, "The randomization technique as a modeling tool and solution procedure for transient Markov processes", *Operations Research*, vol. 32, no. 2, pp. 343–361, 1984.

[34] P. J. Haas and G. S. Shedler, "Regenerative simulation of stochastic Petri nets", in *Proc. International Workshop on Timed Petri Nets*, pp. 14–21, Torino, Italy, July 1985.

[35] P. Heidelberger and A. Goyal, Sensitivity analysis of continuous time Markov chains using uniformization, in *Computer Performance and Reliability*, pp. 93–104, Amsterdam: North-Holland, 1988.

[36] P. Heidelberger and S.S. Lavenberg, "Computer performance evaluation methodology", *IEEE Trans. on Computers*, vol. C-33, no. 12, pp. 1195–1220, December 1984.

[37] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Reading: Addison-Wesley, 1979.

[38] R. A. Howard, *Dynamic Probabilistic Systems, Vol II: Semi-Markov and Decision Processes*, New York: Wiley, 1971.

[39] IEEE, *Token-Passing Bus Access Method, Std 802.4-1985*, New York: IEEE Press, 1985.

[40] B. R. Iyer, L. Donatiello, and P. Heidelberger, "Analysis of performability for stochastic models of fault-tolerant systems", *IEEE Trans. on Computers*, vol. C-35, no. 10, pp. 902–907, October 1986.

[41] J. R. Jackson, "Networks of waiting lines", *Operations Research*, vol. 5, no. 4, pp. 518–521, 1957.

[42] D. Janetzky and K. S. Watson, Performance evaluation of the MAP token bus in real time applications, in *Advances in Local Area Networks*, pp. 411–425, New York: IEEE Press, 1987.

[43] R. M. Keller, "Formal verification of parallel programs", *CACM*, vol. 19, pp. 371–384, July 1976.

[44] J. C. Kemeny and J. L. Snell, *Finite Markov Chains*, Princeton: D. Van Nostrand Co., Inc., 1969.

[45] L. Kleinrock, *Queueing Systems, Volume I: Theory*, New York: John Wiley, 1975.

[46] Z. Kohavi, *Switching and Finite Automata Theory*, New York: McGraw-Hill, 1978.

[47] V. G. Kulkarni, V. F. Nicola, R. M. Smith, and K. S. Trivedi, "Numerical evaluation of performability measures and job completion time in repairable fault-tolerant systems", in *Proc. 16th International Symp. on Fault-Tolerant Computing*, pp. 252–257, Vienna, Austria, July 1986.

[48] J. C. Laprie, "Dependable computing and fault tolerance: Concepts and terminology", in *Proc. 15th International Symp. on Fault-Tolerant Computing*, pp. 2–11, Ann Arbor, MI, June 1985.

[49] S. S. Lavenberg, *Computer Performance Modeling Handbook*, New York: Academic Press, 1983.

[50] J. D. C. Little, "A Proof of the Queueing Formula $L = \lambda W$", *Operations Research*, vol. 9, pp. 383–387, 1961.

[51] MAP Task Force, *Manufacturing Automation Protocol, Version 2.1*, General Motors Technical Center, Warren, MI, 1985.

[52] R. A. Marie, A. L. Reibman, and K. S. Trivedi, "Transient analysis of acyclic Markov chains", *Performance Evaluation*, vol. 7, no. 3, pp. 175–194, August 1987.

[53] M. Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani, "On Petri nets with stochastic timing", in *Proc. International Workshop on Timed Petri Nets*, pp. 80–87, Torino, Italy, July 1985.

[54] M. A. Marsan, G. Balbo, G. Chiola, and S. Donatelli, "On the product-form solution of a class of multiple-bus multiprocessor system models", *Journal of Systems and Software*, vol. 1, no. 2, pp. 117–124, 1986.

[55] M. A. Marsan, G. Balbo, G. Ciardo, and G. Conte, "A software tool for the automatic analysis of generalized stochastic Petri net models.", in *Proc. International Conf. of Modelling Techniques and Tools for Performance Analysis*, Paris, France, May 1984.

[56] M. A. Marsan, G. Balbo, and G. Conte, "A class of generalized stochastic Petri nets for performance evaluation of multiprocessor systems", in *Proc. ACM/SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pp. 198–199, Minn., MN, August 1983.

[57] M. A. Marsan, G. Balbo, and G. Conte, "A class of generalized stochastic Petri nets for performance evaluation of multiprocessor systems", *ACM Trans. on Computer Systems*, vol. 2, no. 2, pp. 93–122, May 1984.

[58] M. A. Marsan, A. Bobbio, G. Conte, and A. Cumani, "Performance analysis of degradable multiprocessor systems using generalized stochastic Petri nets", *IEEE Dist. Processing TC Newsletter*, vol. 6, no. SI-1, pp. 47–54, January 1984.

[59] M. A. Marsan and G. Chiola, "On Petri nets with deterministic and exponential transition firing times", in *Proc. Seventh European Workshop on Application and Theory of Petri Nets*, pp. 151–165, Oxford, June 1986.

[60] M. A. Marsan, G. Chiola, and A. Fumagalli, "An accurate performance model of CSMA/CD bus LAN", in *Proc. Seventh European Workshop on Application and Theory of Petri Nets*, pp. 167–182, Oxford, June 1986.

[61] J. Meyer and L. Wei, "Influence of workload on error recovery in random access memories", *IEEE Trans. on Computers*, vol. 37, no. 4, pp. 500–507, April 1988.

[62] J. F. Meyer, "Closed-form solutions of performability", *IEEE Trans. on Computers*, vol. C-31, pp. 648–657, July 1982.

[63] J. F. Meyer, "On evaluating the performability of degradable computing systems", in *Proc. 8th International Symp. on Fault-Tolerant Computing*, pp. 44–49, Toulouse, France, June 1978.

[64] J. F. Meyer, "On evaluating the performability of degradable computing systems", *IEEE Trans. on Computers*, vol. C-22, pp. 720–731, August 1980.

[65] J. F. Meyer, Performability modeling of distributed real-time systems, in *Mathematical Computer Performance and Reliability*, Amsterdam: North-Holland, 1984.

[66] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application", in *Proc. International Workshop on Timed Petri Nets*, pp. 106–115, Torino, Italy, July 1985.

[67] D. R. Miller, "Reliability calculation using randomization for Markovian fault-tolerant computing systems", in *Proc. 13th International Symp. on Fault-Tolerant Computing*, pp. 284–289, Milano, Italy, June 1983.

[68] C. Moler and C. Van Loan, "Nineteen dubious ways to compute the exponential of a matrix", *SIAM Review*, vol. 20, no. 4, pp. 801–836, October 1978.

[69] M. Molloy, *On the integration of delay and throughput measures in distributed processing models*, PhD thesis, UCLA, 1981.

[70] A. Movaghar, *Performability modeling with stochastic activity networks*, PhD thesis, University of Michigan, 1985.

[71] A. Movaghar and J. F. Meyer, "Performability modeling with stochastic activity networks", in *Proc. 1984 Real-Time Systems Symp.*, Austin, TX, December 1984.

[72] K. H. Muralidhar and J. R. Pimentel, "Performability analysis of the token bus protocol", in *Proc. IEEE INFOCOM '87*, San Francisco, March 1987.

[73] S. Natkin, *Reseaux de Petri stochastiques*, PhD thesis, CNAM-PARIS, 1980.

[74] M. Neuts, *Matrix-geometric solutions in stochastic models*, Baltimore: Johns Hopkins Univ. Press, 1981.

[75] O. Osterby and Z. Zlatev, *Lecture Notes in Computer Science: Direct Methods for Sparse Matrices*, Heidelberg: Springer-Verlag, 1983.

[76] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs: Prentice-Hall, 1981.

[77] L. A. Prisgrove and G. S. Shedler, "Symmetric stochastic Petri nets", *IBM J. Res. Develop.*, vol. 30, no. 3, pp. 278–293, May 1986.

[78] W. Reisig, *Petri Nets*, Berlin: Springer-Verlag, 1982.

[79] M. Rosenblatt, *Markov processes. Structure and Asymptotic Behavior*, Berlin: Springer-Verlag, 1971.

[80] S. R. Sachs, K. Kan, and J.A. Silvester, "Performance analysis of a token-bus protocol and comparisons with other LAN protocols", in *Proc. 10th IEEE Conference on Local Computer Networks*, pp. 100–106, Minneapolis, MN, October 1985.

[81] W. H. Sanders and J. F. Meyer, "METASAN: A performability evaluation tool based on stochastic activity networks", in *Proc. ACM-IEEE Comp. Soc. 1986 Fall Joint Comp. Conf.*, Dallas, TX, November 1986.

[82] S. Shapiro, "A stochastic Petri net with application to modeling occupancy times for concurrent task systems", *Networks*, vol. 9, pp. 375–379, 1979.

[83] R. M. Smith, *Markov reward models: Application domains and solution methods*, PhD thesis, Department of Computer Science, Duke University, 1987.

[84] R. M. Smith and K. S. Trivedi, "A performability analysis of two multi-processor systems", in *Proc. 17th Intenational Symp. on Fault-Tolerant Computing*, Pittsburgh, PA, July 1987.

[85] P. S. Thiagarajan, Elementary net systems, in *Lecture Notes in Computer Science 254*, pp. 26–59, Berlin: Springer-Verlag, 1987.

[86] A. A. Törn, "Simulation nets, a simulation modeling and validation tool", *Simulation*, vol. 45, no. 2, pp. 71–75, August 1985.

[87] J. P. Tremblay and R. Manohar, *Discrete Matematical Structures with Applications to Computer Science*, New York: McGraw-Hill, 1975.

[88] K.S. Trivedi, *Probability & Statistics with Reliability, Queueing and Computer Science Applications*, Englewood Cliffs: Prentice-Hall, 1982.

[89] L. T. Wu, *Models for evaluating the performability of degradable computing systems*, PhD thesis, Univ. of Michigan, 1982.

[90] L.T. Wu, "Operational models for the evaluation of degradable computing systems", in *Proc. ACM/SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pp. 179–185, Seattle, WA, August 1982.

[91] A. Zenie, "Colored stochastic Petri nets", in *Proc. International Workshop on Timed Petri Nets*, pp. 262–271, Torino, Italy, July 1985.

[92] W. M. Zuberek, "Performance evaluation using extended timed Petri nets", in *Proc. International Workshop on Timed Petri Nets*, pp. 272–278, Torino, Italy, July 1985.