THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

UNIFIED PERFORMANCE-RELIABILITY EVALUATION

J.F. Meyer

CRL-TR-27-84

April 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]This work was supported in part by the Communications and Network Laboratory of the Industrial Technology Institute. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the funding agencies.

To appear in ***Proceedings of 1984 American Control Conference***, San Diego, CA, June 1984.

UNIFIED PERFORMANCE-RELIABILITY EVALUATION

J. F. Meyer

ABSTRACT

Unified performance-reliability (performability) evaluation is reviewed relative to needs imposed by distributed real-time systems. Such systems typically exhibit properties of concurrency, timeliness, fault tolerance, and degradable performance, calling for unification and generalization of modeling techniques employed in strict evaluations of performance and reliability. An overview of our work in this area is presented, including a discussion of directions currently being pursued.

1. INTRODUCTION

This paper concerns the development of modeling methods and tools for unified performance-reliability evaluation of distributed real-time systems. The types of models considered are principally stochastic models of an analytic nature. However, nonprobabilistic models can play a useful supporting role, as described in Section 3. The intent of this presentation is to provide an overview of work in this area that has been underway at The University of Michigan since 1976. Also discussed are directions we are taking in our current research, both at The University of Michigan and at the Industrial Technology Institute.

Generally, an evaluation of a system seeks to relate and quantify aspects of what the system is and does with respect to what the system is required to be and do. Accordingly, performance-reliability evaluation methods for distributed real-time systems need to be responsive to the kinds of performance-reliability requirements that accompany the use of such systems. Although these requirements differ in detail from application to application, they have some typical properties which, in turn, impose special conditions on system structure and behavior.

In particular, such requirements typically call for high performance when the system is fault-free. This, coupled with the fact that resources are distributed, results in systems which exploit concurrent (parallel) processing. Secondly, real-time aspects of such requirements often impose conditions on the "timeliness" of processing activities, e.g., requirements for maximum allowable (input-output) response times which impose deadlines on the times that certain processes can be initiated or completed. Thirdly, real-time applications are typified by requirements for high reliability as well as high performance, resulting in fault-tolerant systems which employ added resources and processes for the purpose of tolerating specified types of faults. Finally, due to a balance of demands for both performance and reliability, such requirements typically call for systems which exhibit degradable performance in the presence of faults, i.e., between the extremes of nondegraded performance (as would be exhibited if the system were fault-free throughout utilization) and fully degraded performance (system failure), the system is able to perform at intermediate levels which provide some benefit to the user.

Each of the properties identified above, i.e., 1) *concurrency,* 2) *timeliness,* 3) *fault tolerance,* and 4) *degradable performance,* can influence a system's ability to perform and, hence, each property should be considered in the modeling process. As surveyed in the section that follows, most of the work to date on unified performance-reliability modeling has focused on properties of fault tolerance and degradable performance. Recently, however, efforts are being made to develop models which can accommodate all four properties. This work is discussed in Section 3.

2. PERFORMABILITY MODELING

Traditionally, issues of computing system performance and reliability have been distinguished by regarding "performance" as "how well the system performs, provided it is correct" [1-4] and regarding "reliability" as "the probability of performing successfully" [5-8]. However, if performance is degradable (in which case we say the system itself is degradable) then a performance evaluation (of the correct or fault-free system) will generally not suffice since structural changes, due to faults, may be the cause of degraded performance. By the same token, an evaluation of reliability (or of reliability related measures such as availability and maintainability) will generally not suffice since "success" can take on various meanings and, in particular, it need not be identified with "absence of system failure."

What is called for instead are evaluations with respect to unified performance-reliability measures which, in our terminology [9,10], quantify a system's *performability.* Such measures, in turn, call for appropriate generalizations of the types of models and solution methods employed in strict evaluations of either performance or reliability.

To accommodate these needs, a general modeling framework was introduced in [9] (and subsequently refined in [10]) for evaluating the performability of a system S. (Generally, S is interpreted as including not only a system, per se, but also relevant aspects of its environment that impact its ability to perform.) In this framework, 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 [9,10]) induced by Y where, for any measurable set B of accomplishment levels ($B \subseteq A$), $Perf(B)$ is the probability that S 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. A base model X together with a performance variable Y is an analytic ***performability model*** of S. (We are omitting some details here regarding how Y must relate to X, resulting in a slight departure from the definition given in [10].) Performability model ***construction*** is the process of identifying a performance variable Y and determining a base model stochastic process X that permits a 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. Generally, knowledge of the probability distribution function (PDF) of Y suffices to determine such values. Accordingly, we regard an analytic performability model as "solved" once the PDF of Y is obtained. Solution of a PDF can be either a closed-form solution (expressed as a function of base model and performance variable parameters) or a numerical solution.

The concept of performability is thus quite general and, depending on the choice of base model X and performance variable Y, can be specialized to usual notions of (strict) performance, reliability, and reliability related concepts such as availability and maintainability. For example, if the structure of a computing system S is presumed to be fixed and correct (i.e., X represents changes only in S's internal state and environment) and, for a designated utilization period $T = [0,t]$, we let Y be the average throughput rate during $[0,t]$, i.e.,

$$Y = \frac{\text{no. of jobs processed during}[0,t]}{t}$$

then (X,Y) is a (strict) performance model. At the other extreme, if the states of X can be classified as either operational (corresponding to a working system) or nonoperational (a failed system) and we let

$$Y = \begin{cases} 1 \text{ if } X \text{ is in an operational} \\ \quad \text{state throughout } [0,t] \,, \\ \\ 0 \text{ else} \end{cases}$$

then (X,Y) is a reliability model. Generally, however, a performability model is neither a performance model nor a reliability model. Moreover, experience has shown that such models are sufficiently general to quantify a degradable system's ability to perform in the presence of faults.

At The University of Michigan, our work on the formulation, development, and application of (model-based) performability evaluation methods began in 1976 under a grant (NSG 1306) from the NASA Langley Research Center. The application focus of this research was the type of ultrareliable aircraft control computers being developed for Langley by both Stanford Research Institute Int'l (SIFT; [11]) and the C.S. Draper Laboratory (FTMP; [12]). This grant continued for a period of six years, resulting in definition of the basic modeling framework [9,10] and a variety of studies concerning performability evaluation methods, tools, and applications [13-20]. (Also see the doctoral theses of Wu [21] and Furchtgott [22] for more detailed descriptions of much of this work.) Since 1982, our work has been aimed at the development of more generally applicable solution methods [23,24] and more refined methods of model construction that can capture (in a single model) properties of concurrency, timeliness, fault tolerance, and degradable performance [25]. (The latter effort is reviewed in Section 3.)

The need for unified performance-reliability evaluation methods has likewise been recognized by others who, with various approaches, have contributed to the basic literature on this emerging discipline. This includes studies by Beaudry ([26,27]; performance-related reliability measures), Losq ([28]; degradable systems composed of degradable resources), Troy ([29]; efficiency evaluation), Gay and Ketelsen ([30]; performance evaluation of degradable systems), Mine and Hatayama ([31]; job-related reliability), De Souza ([32]; benefit analysis of fault tolerance), Castillo and Siewiorek ([33,34]; performance-reliability models for computing systems), Chou and Abraham ([35]; performance-availability models of shared resource multiprocessors). Osaki and Nishio ([36]; reliability of information), Beyaert, Florin, Lonc, and Natkin ([37]; dependability evaluation using stochastic Petri nets), Huslende ([38,39]; combined performance-reliability evaluation for degradable systems), Arlat and Laprie ([40]; performance-related dependability evaluation), Munarin ([41]; performance/reliability analysis of gracefully degrading systems), and Krishna and Shin ([42]; performance measures for multiprocessor controllers).

Our work in this area can be broadly classified according to the type of performability model employed. A basic distinction in this regard is whether the performance variable Y is discrete (its range is finite or denumerable) or continuous. In the case of a ***discrete performance variable*** (DPV) and, more specifically, when Y is finite-valued, the performability model construction and solution methods we've developed can be viewed, for the most part, as generalizations of reliability modeling techniques. In the case of a ***continuous performance variable*** (CPV), on the other hand, most of the methods we've developed or are currently investigating derive from an appropriate blend of both reliability modeling and (strict) performance modeling techniques. In the discussion that follows, methods and tools for the construction and solution of DPV models are referred to simply as DPV methods and tools. Similar terminology applies in the case of CPV models.

2.1. DPV Methods and Tools

Much of the work under the above referenced NASA grant was concerned with the development of DPV methods and tools. This was motivated by NASA's interest in user-oriented performability evaluations of fault-tolerant digital control computers for (next generation) commercial aircraft. The user in this case is assumed to be an airline that owns the aircraft and performance (as modeled by Y) is taken to be the quality of an aircraft flight controlled by the computer. With this view of performance, a relatively small number of accomplishment levels (possible values of Y) will suffice, resulting in a DPV model of the system. More generally, user-oriented (sometimes referred to as "mission-oriented") evaluation problems of this type are naturally suited to DPV's and, indeed, to finite-valued performance variables where the number of accomplishment levels is at least 3 but typically no more than 10. (In the case of only 2 accomplishment levels, interpreted as "success" and "failure", performance is no longer degradable and performability reduces to reliability.)

Construction and solution of DPV performability models exploit a well known property of discrete random variables which, in the terminology of the modeling framework, says the following. Since Y is a DPV, the set A of possible accomplishment levels is countable and, hence, the probability distribution

$$\{P[Y = a] | a \in A\}$$

suffices to determine the performability of S. Accordingly, in the case of a DPV model, the performability of S can be alternatively defined as the function $Perf : A \rightarrow [0,1]$ where, for $a \in A$,

$$Perf(a) = P[Y = a] .$$

If we assume further that A is finite then performability is likewise finite, i.e., the values of $Perf$ can be tabulated.

The base model part (the stochastic process X) of a DPV model represents a sufficiently detailed probabilistic description of changes in a system's structure (due to faults and fault recovery), internal state, and environment to permit solution of the performability values $Perf(a)$. Generally, a base model (for DPV's) is a continuous-time, finite-state stochastic process which is constructed by first decomposing the utilization period $[0,t]$ into a finite number of consecutive time periods called ***phases***. For each phase, a system's intraphase behavior is represented by a (continuous-time, finite-state) stochastic process which is typically a time-homogeneous Markov process. Different phases, however, may be modeled by different processes, subject to constraints which permit the determination of (conditional) interphase transition probabilities. Combination of these processes results in a ***phased model*** [15] which serves as the base model X. Relative to a specified performance variable Y, X is required to "support" Y in the sense that the end-of-phase samples of X uniquely determine the value of Y.

DPV performability evaluation can thus be viewed as a generalization of "phased mission" reliability evaluation (see [43-45], for example). Techniques used to construct and solve DPV models are likewise more general and complex, due in part to the less restricted nature of phased base models. A more challenging property of DPV models, however, is due to the general manner in which the base model X relates to the performance variable Y, via a function referred to as the ***capability function*** [9,10]. Relative to a given level of accomplishment, the end-of-phase samples of X can be ***functionally dependent*** [13], e.g., knowing that $Y = a$, knowledge of the state of X at the end of phase i can contribute to knowledge of the state of X at the end of some other phase j. In contrast, assumptions made in the construction of phased mission reliability models (see [45], for example) are such that, with respect to the accomplishment level "success," the phases must be functionally independent (as established in [13], Theorem 6).

Given a DPV model (X,Y) and an accomplishment level a, solution of the performability value $Perf(a)$ is, conceptually, a two-step procedure:

1) Determine the set U_a of all base model state trajectories (as characterized by their end-of-phase samples) that correspond, via the capability function, to $Y = a$.

2) Using knowledge of X, determine the probability of the trajectory set U_a. This probability is the value of $Perf(a)$.

Execution of step 1) exploits a hierarchical elaboration of the base model and, using matrix representations of trajectory sets and an accompanying matrix calculus, proceeds, in a top-down manner, to determine a suitable representation of U_a (see [10,22]). An important feature of this method is the ability to account for functional dependencies in the process of determining U_a. As a consequence, U_a is ultimately represented as a union of disjoint "Cartesian" trajectory sets. Other approaches require such dependencies to be recognized, manually, during the process of model construction (see [46] for a detailed assessment of these differences).

Step 2) is performed by first calculating the probability of each of the Cartesian subsets determined in step 1), using formulas described in [15,21] and developed for this purpose. Since these subsets are disjoint, the sum of these probabilities yields the desired solution.

Each of these steps can involve a considerable amount of symbol manipulation and numerical computation, particularly in the case of a relatively complex degradable system. This, together with problems encountered in model construction, calls for programmed performability evaluation tools that can aid both the construction and solution phases of the evaluation process. Over the past several years, considerable effort has been devoted to the development of such tools which, collectively, reside in a software package called **METAPHOR** (**M**ichigan **E**valua**T**ion **A**id for **P**erp**HOR**mability). (In addition to DPV tools, METAPHOR also houses tools for the construction and solution of CPV models). The current version of METAPHOR is written in *C* and runs under UNIX. The portion devoted to DPV methods, called "meta_discrete" , contains approximately 8,000 lines of source code and has approximately 256 Kbytes of executable code. Descriptions of various parts of METAPHOR (to the extent that they are currently documented) can be found in [22].

Applications of DPV methods and tools are reported in a number of papers and technical reports [14,16-18,21,22,46]. These include a relatively comprehensive performability evaluation of the SIFT computer [16,17] where SIFT's computational environment is a transoceanic flight of an advanced commercial aircraft. Performability, in this case, is quantified in terms of 5 accomplishment levels, ranging from a "perfect" flight to loss of the aircraft. Details concerning the use of METAPHOR in this and other applications appear in [22].

Generally, DPV models are best suited to performability evaluations where the user (of the evaluation results) is interested in whether a system satisfies certain "bottom line" performability requirements. However, if the evaluation results disclose that the system is deficient, these results may not be refined enough to indicate how the system design should be modified. Design-oriented performability evaluation thus calls for lower level, more detailed views of a system's productivity, responsiveness, etc. which are more naturally represented by continuous performance variables (CPV's).

2.2. CPV Methods and Tools

When system performance ranges over a continuum of accomplishment levels, the construction and solution of performability models is complicated by the need to deal with a much greater amount of information. In constructing the base model, a more detailed representation of the system's internal state and environment is typically required so that the base model can support the performance variable. On the solution side, performability is determined by the probability distribution function (PDF) of the performance variable (conceptually an uncountable set) as opposed to the kind of discrete distributions associated with DPV's.

Generally, the types of CPV models we've investigated derive from concepts and constructs used in both performance modeling and reliability modeling. Accordingly, a base model will typically represent activities relating to performance (e.g., job arrivals, job processing) as well as activities relating to reliability (e.g., fault arrivals). However, a fundamental difficulty encountered in solving CPV performability models is due to the fact that performance related activities usually occur at much higher rates than do reliability related activities. Hence, in the case of a Markovian base model, the ratio between the highest and lowest state transition-rate can be many orders of magnitude. In other words, the differential equations describing the model's state behavior (i.e., those determined by the generator matrix of the Markov model) are "stiff" (see [47], for example) and, when solved numerically, require special treatment to avoid excessive errors.

Various approaches can be used to deal with this stiffness property and all deserve more extensive investigation. One approach is to exploit numerical solution methods developed specifically for stiff equations, e.g., Gear's methods [47]. A second approach is to find alternative formulations of the state occupancy probabilities such that solutions are less subject to numerical errors. An example of this approach are formulas obtained via "selective randomization" [48] where the Markov model is "randomized" with respect to states having high transition rates.

A third approach, and one that we have exploited [19,20], is to lump states with high transition rates such that the lumped model is no longer stiff. Performance rates in the lumped states are determined via steady-state solution techniques, under the assumption that the consequences of high rate activities approach equilibrium conditions between the completions of low rate activities. (As a result of this assumption, the solution obtained is approximate; it appears to be a good approximation, however, particularly in the case of very stiff base models). Once these performance rates are determined, performability is solved in terms of the lumped model. The latter step is far from trivial (see [20], Section III) but is capable of producing closed-form solutions as well as numerical solutions.

Since the lumped model referred to above represents changes in the system's structure (due to faults), an equivalent approach is to regard the lumped model as the base model X. States of X are interpreted as "structure states" of S and hence X is a finite-state stochastic process. With this approach, however, we do not require that X be Markovian or even semi-Markovian. Performance aspects are then accounted for by associating a performance rate with each structure state. The stochastic process X together with the performance rates can thus be viewed as a ***reward model*** (see [49], for example) where the performance rates become reward rates. Accordingly, the performance variable Y is taken to be the total reward accrued during a specified utilization period $[0,t]$.

CPV models of this type constitute a broad class of performability models with widespread applicability. Of particular interest is their use in modeling systems which, during $[0,t]$, are assumed to be "nonrepairable" (e.g., t is the period between successive instances of scheduled periodic maintenance). In this case the base model process X is ***acyclic,*** i.e., the probability of entering the same (structure) state more than once is 0. If we assume further that the reward rates are such that the reward model is ***nonrecoverable*** [21], i.e., reward rates experienced cannot increase with time, such models admit to a general solution method described in [23,24]. Specifically, if (X,Y) is a performability model satisfying the above conditions, this method determines the PDF $F_Y(y)$ of Y and, hence, the performability of the modeled system. The method involves expression of $F_Y(y)$ as a sum of definite integrals which, if solved symbolically, yields a closed-form solution. Alternatively, for a given assignment of model parameter values, this integral expression can be solved numerically for various values of y, resulting in a numerical solution of $F_Y(y)$. Details concerning development and justification of the method can be found in [22, Chap.5].

Due to the inherent complexity of most CPV models, implementation of this technique for either closed-form or numerical solutions requires the support of programmed evaluation tools. This is particularly so in the case of numerical solutions, since these are sought when the model is too complex to admit a meaningful closed-form solution. Development of a computer program for this purpose has recently been completed and incorporated in the CPV portion of METAPHOR, referred to as "meta_continuous" [22]. Meta_continuous contains approximately 4,000 lines of C code, including menu and project management functions.

Applications of CPV methods and tools have been demonstrated for various types of degradable, nonrepairable systems, beginning with the dual-processor example considered in [19,20]. In this example, performance is taken to be the system's (normalized) average throughput rate; degradable performance results from the system's ability to recover (with a specified "coverage") from a processor fault. Using the lumping technique outlined above, the base model is simple enough to permit manual derivation of a closed-form solution of performability. More complex systems involving multiple (more than 2) processors have been evaluated [22,23] using the methods and tools described in the preceding three paragraphs. These include a multiprocessor/air conditioner example where effects of a faulty air conditioner are such that processor failure rates start to vary as a function of time. The base model, in this case, is not semi-Markov and yet a performability solution using meta_continuous can still be obtained.

3. CURRENT RESEARCH

Most of the work to date on unified performance-reliability evaluation has focused on accommodating properties of fault tolerance and degradable performance. However, in the case of distributed real-time systems, properties of concurrency and timeliness can likewise influence a system's ability to perform. Accordingly, when such evaluations are model-based (on either analytic models or simulation models), all four properties need to be dealt with effectively in the modeling process.

During the past 20 years, considerable effort has been devoted to the modeling of concurrent systems for the purpose of behavior analysis and, to a lesser extent, performance evaluation. Much of this activity has been directed toward behavior analysis in a nonprobabilistic setting through the use of Petri nets, derivatives thereof, and other model types which are behaviorally equivalent to Petri nets (see [50] for a comprehensive discussion of such models). Efforts have also been made to extend Petri-type nets, via the introduction of timing, to obtain models that are better suited to performance evaluation [51-56]. Regarding probabilistic models that deal with concurrency, one might legitimately include queueing networks (as surveyed, for example, in [57]) but, typically, such models treat concurrency at much higher (less detailed) levels than do Petri-type models. Of greater relevance are probabilistic network models that capture concurrency at lower levels. The latter include GERT Networks [58] and General Activity Networks [59] that date back to 1964, along with more recent models which are direct probabilistic extensions of Petri nets [37,60,61,62].

Most of the models referred to in the previous paragraph presume a fixed system structure, thus excluding considerations of fault tolerance and degradable performance. A notable exception is the work of Beyaert, Florin, Lonc, and Natkin [37] on stochastic Petri nets wherein the occurrence of a fault is represented by the firing of a Petri net transition. To the extent that queueing models are able to represent concurrency, other exceptions are the work of Gay and Ketelsen [30], Huslende [38,39] and ourselves [19,20].

Accordingly, a natural direction to pursue is the development of new model classes that incorporate features of both queueing networks and stochastic Petri nets. Moreover, when defining such models, we believe it is helpful to first define a class of models which are nondeterministic (in their state behavior) but are not probabilistic. Via a specified interpretation of how states change with time, each nondeterministic model, when augmented by a designated set of probability distributions, yields a corresponding probabilistic model. Although model-based evaluations of performability call for probabilistic models, the formulation of such models can thus employ nonprobabilistic models of the type described above. Moreover, these nonprobabilistic "skeletons" might also be used directly to verify certain model properties. One advantage of this two-step definition method is that it decomposes a relatively complex concept into simpler, more understandable parts. A much more important advantage, however, is that it facilitates the establishment of explicit connections between the structure/behavior of nondeterministic models and that of their corresponding probabilistic extensions.

Work in this direction has been undertaken at The University of Michigan (Computing Research Laboratory) and the Industrial Technology Institute (Communications and Network Laboratory) with the following consequences. As the first step in a two-step definition (see above) and at the network level of abstraction, we have defined a class of nondeterministic models called ***activity networks***. These networks are more general than Petri nets, where the need to generalize is due to the following important observation. Petri nets (and most other nonprobabilistic models of concurrent systems) exhibit nondeterministic behavior as the consequence of ***temporal uncertainty,*** i.e., among a set of concurrent activities, there is uncertainty as to which activity will be completed first. In other words (in Petri net terms), among a set of enabled transitions, there is uncertainty as to which transition will fire. Moreover, this is the only source of nondeterminism since enabled transitions are uniquely determined by the state (marking) of a Petri net and the next state is uniquely determined by the present state and the transition that fires.

However, when modeling the structure and behavior of complex systems, one wants to represent ***spatial uncertainty*** as well as temporal uncertainty, e.g., uncertainty about which activities are initiated in a given state or, on the completion of an activity, uncertainty about the next state of the system. Such uncertainty is often related to faults, e.g., if the activity in question is a fault recovery process, completion of this activity may result in one of several states, including a state that represents system failure. Spatial uncertainty also occurs in representations of fault-free behavior. In a queueing network, for example, when service of a customer is completed (in some node of the network) the customer may leave the system or proceed to one of several nodes specified by the interconnections of the network. In a queueing network model, this uncertainty is quantified by an assignment of branching probabilities to the output connections of each node.

In our formal definition of an activity network, the structural primitives are ***places*** (as in Petri nets), ***activities*** (which are similar to Petri net transitions, but are augmented by ***cases***), and ***gates***. Activities are of two kinds: ***timed*** activities and ***instantaneous*** activities. 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 associated with that activity. Gates are introduced to permit greater flexibility in defining enabling and completion rules.

To obtain a corresponding class of probabilistic models (the second step of the two-step definition process), we extend the definition of an activity network to that of ***stochastic activity network***. This is done by quantifying spatial nondeterminism with probabilities and by quantifying temporal uncertainty with probability distribution functions for a subclass of activity networks that are ***well behaved***. More specifically, to a given well behaved activity network N with some specified initial marking we adjoin functions C and F, where C specifies the probability distributions of case selections and F specifies the probability distribution functions of activity times. These distributions generally depend on the marking of the network as well as its structure, affording us a great deal of flexibility in the model construction process.

To assist the process of describing activity network and stochastic activity network behavior, two additional model classes are distinguished at the state-transition level: ***activity systems*** and ***stochastic activity systems***. Activity systems are not new (see the "named transition systems" of [63,64], for example) and are abstract enough to represent a large variety of nonprobabilistic systems. In particular, they provide a natural, higher level representation of activity networks. The most detailed description of activity system behavior are its ***state-activity*** sequences, i.e., for a given initial state, the possible sequences of alternating states and activities that can result from a finite number of applications of the transition relation. Thus, when an activity system represents an activity network, the source of spatial uncertainty (completion vs. enabling) is no longer distinguished.

Stochastic activity systems are probabilistic extensions of activity systems where the extension is similar to that made at the network level. Moreover, in the manner that activity systems represent activity networks, stochastic activity systems are higher level representations of stochastic activity networks. The state behavior of a stochastic activity system is a stochastic process which can serve as the base model of a performability model. More precisely, suppose S is a system modeled by a stochastic activity network N. If M is the stochastic activity system corresponding to N then the state behavior of M is a base model X of S. X together with a designated performance variable Y comprise a performability model of S.

We are currently investigating properties of stochastic activity networks (SAN's) and their corresponding stochastic activity systems (SAS's), where this effort has two purposes. One purpose is to determine the general capabilities and limitations of these models so as to compare them with queueing networks and stochastic Petri nets. The second purpose is to obtain a clear understanding of structure-behavior relations for particular model types, providing knowledge that can facilitate the construction and solution of specific performability models. For example, if the model is a SAN we would like to know conditions under which its corresponding SAS has some specified property, e.g., its state behavior is a finite-state process, a Markov process, a semi-Markov process, etc.

We are also pursuing the development of performability model construction methods that incorporate these model classes. Model construction, in this context, is generally viewed as follows. Given a system S and a designated performance variable Y, the first step is to construct an activity network model of S which is refined enough to support an eventual solution of the system's performability (with respect to Y). Also, this network should be sufficiently detailed to permit specification of the probability distributions that convert it to a stochastic activity network (SAN). Since a model at this level may be too complex to admit a solution, there is a need for aggregation methods that can produce simpler SAN models. At some appropriate level of detail, one determines the SAN's corresponding SAS and attempts to characterize its state behavior. The latter may call for additional simplification methods applicable to SAS's. The construction procedure terminates with a stochastic process that can serve as a base model for the subsequent solution phase.

Implementation of this procedure is currently underway at the Industrial Technology Institute. In particular, we have initiated work on a computer program (to be incorporated in METAPHOR) which, when provided with a structural description of a SAN, will automatically derive its corresponding SAS and, specifically, the stochastic process identified with its state behavior.

4. REFERENCES

[1] L. Kleinrock, *Queuing Systems, Volume II: Computer Applications.* New York: John Wiley, 1976.

[2] D. Ferrari, *Computer Systems Performance Evaluation.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

[3] H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Reading, MA: Addison-Wesley, 1978.

[4] C.H. Sauer and K.M. Chandy, *Computer Systems Performance Modeling.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[5] W.G. Bouricius, W.C. Carter, and P.R. Schneider, "Reliability modeling techniques for self-repairing computer systems," in *Proc. 24th ACM Nat. Conf.,* Aug. 1969, pp.295-305.

[6] J.C. Laprie, "Reliability and availability of repairable structures," in *Proc. 1975 Int. Symp. on Fault-Tolerant Computing,* Paris, France, June 1975, pp.87-92.

[7] Y.W. Ng and A. Avizienis, "A reliability model for gracefully degrading and repairable fault-tolerant systems," in *Proc. 1977 Int. Symp. on Fault-Tolerant*

Computing, Los Angeles, CA, June 1977, pp.22-28.

[8] A. Costes, C. Landrault, and J.C. Laprie, "Reliability and availability models for maintained systems featuring hardware failures and design faults," *IEEE Trans. Comput.,* vol.C-27, pp.548-560, June 1978.

[9] J.F. Meyer, "On evaluating the performability of degradable computing systems," in *Proc. 1978 Int. Symp. on Fault-Tolerant Computing,* Toulouse, France, June 1978, pp.44-49.

[10] J.F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Trans. Comput.,* vol.C-22, pp.720-731, Aug. 1980.

[11] J.H. Wensley, *et al.,* "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. of the IEEE,* vol.66, pp.1240-1255, Oct. 1978.

[12] A. L. Hopkins, *et al.,* "FTMP--A highly reliable fault-tolerant multiprocess for aircraft," *Proc. of the IEEE,* vol.66, pp.1221-1239, Oct. 1978.

[13] R.A. Ballance and J.F. Meyer, "Functional dependence and its application to system evaluation," in *Proc. 1978 Johns Hopkins Conf. on Info. Sci. and Syst.,* Baltimore, MD, March 1978, pp.280-285.

[14] D.G. Furchtgott and J.F. Meyer, "Performability evaluation of fault-tolerant multiprocessors," in *1978 Government Micro-circuit Applications Conf. Digest of Papers,* Monterey, CA, Nov. 1978, pp.362-365.

[15] L.T. Wu and J.F. Meyer, "Phased models for evaluating the performability of computing systems," in *Proc. of the 1979 Johns Hopkins Conf. on Information Sciences and Systems,* Baltimore, MD, March 1979, pp.426-431.

[16] J.F. Meyer, D.G. Furchtgott, and L.T. Wu, "Performability evaluation of the SIFT computer," in *Proc. 1979 Int. Symp. on Fault-Tolerant Computing,* Madison, WI, June 1979, pp.43-50.

[17] J.F. Meyer, D.G. Furchtgott, and L.T. Wu, "Performability evaluation of the SIFT computer," *IEEE Trans. Comput.,* vol.C-22, pp.501-509, June 1980.

[18] J.F. Meyer and L.T. Wu, "Evaluation of computing systems using functionals of a Markov Process," in *Proc. 14th Annu. Hawaii Int. Conf. on Syst. Sci.,* Honolulu, HI, Jan. 1981, pp.74-83.

[19] J.F. Meyer, "Closed-form solutions of performability," in *Proc. 1981 Int. Symp. on Fault-Tolerant Computing,* Portland, ME, June 1981, pp.66-71.

[20] J.F. Meyer, "Closed-form solutions of performability," *IEEE Trans. Comput.,* vol.C-31, pp.648-657, July 1982.

[21] L.T. Wu, "Models for evaluating the performability of degradable computing systems," Tech. Report CRL-TR-7-82, Univ. of Michigan, Ann Arbor, MI, June 1982.

[22] D.G. Furchtgott, "Performability models and solutions," Tech. Report CRL-TR-8-84, Univ. of Michigan, Ann Arbor, MI, Jan. 1984.

[23] D.G. Furchtgott and J.F. Meyer, "Performability evaluation of computing systems using reward models," Tech. Report CRL-TR-27-83, Univ. of Michigan, Ann Arbor, MI, Aug. 1983.

[24] D.G. Furchtgott and J.F. Meyer, "A performability solution method for degradable, nonrepairable systems," *IEEE Trans. Comput.,* vol.C-33, June 1984 (to appear).

[25] J.F. Meyer, "Performability modeling of distributed real-time systems," in *Proc. 1983 Int'l Workshop on Applied Math. and Perf./Reliability Models of Comp./Comm. Sys.*, G. Iazeolla and S. Tucci (eds.), Amsterdam, The Netherlands: North-Holland, 1983.

[26] M.D. Beaudry, "Performance related reliability measures for computing systems," in *Proc. 1977 Int. Symp. on Fault-Tolerant Computing,* Los Angeles, CA, June 1977, pp.16-21.

[27] M.D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Computers,* vol.C-27, pp.540-547, June 1978.

[28] J. Losq, "Effects of failures on gracefully degradable systems," in *Proc. 1977 Int. Symp. on Fault-Tolerant Computing,* Los Angeles, CA, June 1977, pp.29-34.

[29] R. Troy, "Dynamic reconfiguration: An algorithm and its efficiency evaluation," in *Proc. 1977 Int. Symp. on Fault-Tolerant Computing,* Los Angeles, CA, June 1977, pp.44-49.

[30] F.A. Gay and M.L. Ketelsen, "Performance evaluation of gracefully degrading systems," in *Proc. 1979 Int. Symp. on Fault-Tolerant Computing,* Madison, WI, June 1979, pp.51-58.

[31] H. Mine and K. Hatayama, "Performance related reliability measures for computing systems," in *Proc. 1979 Int. Symp. on Fault-Tolerant Computing,* Madison, WI, June 1979, pp.59-62.

[32] J.M. De Souza, "A unified method for the benefit analysis of fault-tolerance," in *Proc. 1980 Int. Symp. on Fault-Tolerant Computing,* Kyoto, Japan, Oct. 1980, pp.201-203.

[33] X. Castillo and D.P. Siewiorek, "A performance reliability model for computing systems," in *Proc. 1980 Int. Symp. on Fault-Tolerant Computing,* Kyoto, Japan, Oct. 1980, pp.187-192.

[34] X. Castillo and D.P. Siewiorek, "Workload, performance, and reliability of digital computing systems," in *Proc. 1981 Int. Symp. on Fault-Tolerant Computing,* Portland, ME, June 1981, pp.84-89.

[35] T.C.K. Chou and J.A. Abraham, "Performance/availability model of shared resource multiprocessors," *IEEE Trans. Reliability,* vol.R-29, no.1, pp.70-76, April 1980.

[36] S. Osaki and T. Nishio, "Reliability evaluation of some fault-tolerant computer architectures," in *Lecture Notes in Computer Science.* Berlin, Germany: Springer-Verlag, 1980.

[37] B. Beyaert, G. Florin, P. Lonc, and S. Natkin, "Evaluation of computer system dependability using stochastic petri nets," in *Proc. 1981 11th Int. Symp. on Fault-Tolerant Computing,* Portland, ME, June 1981, pp.66-71.

[38] E. Huslende, "A combined evaluation of performance and reliability for degradable systems," in *Proc. ACM/SIGMETRICS Conf. on Meas. and Modeling of Computing Syst.,* Las Vegas, Nevada, Sept. 1981, pp.157-164.

[39] E. Huslende, "Optimal/cost reliability allocation in communication networks" in *Proc. 1983 Int. Symp. on Fault-Tolerant Computing,* Milano, Italy, June 1983, pp.348,355.

[40] J. Arlat and J.C. Laprie, "Performance-related dependability evaluation of super-computer systems" in *Proc. 1983 Int. Symp. on Fault-Tolerant Computing,* Milano, Italy, June 1983, pp.276,283.

[41] J.A. Munarin, "Dynamic workload model for performance/reliability analysis of gracefully degrading systems" in *Proc. 1983 Int. Symp. on Fault-Tolerant Computing,* Milano, Italy, June 1983, pp.290-295.

[42] C.M. Krishna and K.G. Shin, "Performance measures for multiprocessor controllers," in *Performance '83.* Agrawala, A. K. and Tripathi, S. K. (eds.), Amsterdam: North-Holland, 1983, pp.229-250.

[43] H.S. Winokur, Jr. and L.J. Goldstein, "Analysis of mission-oriented systems," *IEEE Trans. Reliability,* vol.R-18, no.4, pp.144-148, Nov. 1969.

[44] J.L. Bricker, "A unified method for analyzing mission reliability for fault tolerant computer systems," *IEEE Trans. Reliability,* vol.R-22, no.2, pp.72-77, June 1973.

[45] J.D. Esary and H. Ziehms, "Reliability analysis of phased missions," in *Reliability and Fault Tree Analysis.* Philadelphia, PA: SIAM, 1975, pp.213-236.

[46] E.F. Hitt, M.S. Bridgman, and A.C. Robinson, "Comparative analysis of techniques for evaluating the effectiveness of aircraft computing systems," in *NASA Contractor Report l59358.* Battelle Columbus Laboratories, Columbus, OH, April 1981.

[47] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* Englewood Cliffs, NJ: Prentice-Hall, 1971.

[48] D.R. Miller, "Reliability calculation using randomization for Markovian fault-tolerant computing systems," in *Proc. 1983 Int. Symp. Fault-Tolerant Computing,* Milano, Italy, June 1983, pp.284-289.

[49] R.A. Howard, *Dynamic Probabilistic Systems, Vol. II: Semi-Markov and Decision Processes.* New York, NY: Wiley, 1971.

[50] J.L. Peterson, *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[51] G. Nutt, "The formulation and application of evaluation nets," Ph.D. Thesis, Computer Science Group, Univ. of Washington, July 1972.

[52] C. Ramchandani, "Analysis of asynchronous concurrent systems by Petri nets," Ph.D. Thesis, Dept. of Electrical Engineering, MIT, July 1973.

[53] J.L. Baer and J. Jensen, "Simulation of large parallel systems: Modeling of tasks," in *Measuring, Modeling, and Evaluating Computer Systems.* H. Bellner and E. Gelenbe (eds.), Amsterdam: North-Holland, 1977, pp.53-73.

[54] Y.W. Han, "Performance evaluation of a digital system using a Petri net-like approach," in *Proc. of the National Electronic Conf.,* vol.32, 1978, pp.155-160.

[55] J.D. Noe, "Nets in modeling and simulation," in *Net Theory and Application, Lecture Notes in Computer Science.* Berlin, Germany: Springer-Verlag, 1980.

[56] J. Sifakis, "Performance evaluation of systems using nets," in *Net Theory and Application, Lecture Notes in Computer Science.* Berlin, Germany: Springer-Verlag, 1980.

[57] *Computing Surveys* (Special issue on queuing network models of computer systems performance), vol.10, no.3, Sept. 1978.

UNIVERSITY OF MICHIGAN
3 9015 03483 4096

[58] A.A.B. Pritsker and W.W. Happ, "GERT: Graphical evaluation and review technique - Part I. fundamentals," *Journal of Industrial Engineering,* vol.17, no.5, pp.267-274, May 1966.

[59] S.E. Elmaghraby, "An algebra for the analysis of generalized activity networks," *Management Science,* vol.10, pp.621-631, 1964.

[60] S. Shapiro, "A stochastic Petri net with application to modeling occupancy times for concurrent task systems," *Networks,* vol.9, pp.375-379, 1979.

[61] M. Molloy, "On the integration of delay and throughput measures in distributed processing models," Ph.D. Thesis, UCLA Computer Science Dept., June 1981.

[62] 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,* Minn., MN, August 1983, pp.198-199.

[63] R.M. Keller, "Vector replacement systems: A formalism for modeling asynchronous systems," Tech. Report 117, Computer Science Lab., Princeton Univ., Dec. 1972.

[64] R.M. Keller, "Formal verification of parallel programs," *CACM,* vol.19, pp.371-384, July 1976.