Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

SYSTEMS MODELING WITH XPETRI

Robert Geist
Darren Crane

Department of Computer Science
Clemson University
Clemson, South Carolina 29634-1906, USA

Stephen Daniel
Darrell Suggs

Data General Corporation
62 Alexander Drive
Research Triangle Park, North Carolina 27711, USA

ABSTRACT

The design of a new modeling tool, *xpetri*, is described, and its use in solving real computer system design problems is illustrated. The tool is based on an extension of stochastic Petri nets, and provides both a succinct model language specification and a wide-ranging collection of modeling capabilities including detailed stochastic workload representation. Although solutions of models in the specified *xpetri* language are outside the realm of purely analytic techniques, favorable solution times are provided via a multi-threaded simulation compiler. A technique is suggested for allowing the threads to execute transition firing outside strictly time-sequential order. An X-windows (Motif) interface offers fast and reliable generation of models in the *xpetri* language.

1 INTRODUCTION

Intrinsic limitations on the representation capabilities of queueing network models, together with strong demands for improved accuracy in performance predictions from models, have led to a growing interest in computer systems modeling that is based on *stochastic Petri nets*. Recall that a Petri net is a directed bipartite graph whose two vertex sets are called *places* and *transitions*. Places are traditionally represented by circles and transitions by rectangles. Places may contain one or more *tokens*, represented by small discs. The semantics attached to such nets are rules for simulation:

- If every input place to a transition contains one or more tokens, the transition is *enabled*.
- Enabled transitions may *fire*, that is, remove one token from each input place and add one token to each output place.
- If firing an enabled transition would disable a concurrently enabled transition (conflict), the firing transition is chosen at random.

A common extension to the basic Petri nets, *inhibitor arcs*, adds Turing completeness: If an arc from a place to a transition is an inhibitor arc, the transition is enabled only if the place is empty.

Largely due to the ease with which modelers can represent common system features such as concurrency and resource contention, Petri nets and their extensions (Dugan et al. 1984, Jensen, 1987, Malloy, 1982, Marsan, Conte, and Balbo, 1984) have been used by many authors in computer systems performance modeling (Balbo, Bruell, and Ghanta, 1988, Holliday and Vernon, 1987) and reliability modeling (Shieh, Ghosal, and Tripathi, 1989, Yoneda, Nakade, Tohma, 1989).

Although many extensions to Petri nets have been proposed, few have been widely accepted by the community of industrial computer system designers. We conjecture that one reason for this reluctance is that the increased modeling flexibility is often unmotivated by realistic system design examples, and yet frequently attended by a large increase in specification semantic complexity. A notable exception is the Generalized Stochastic Petri Net (GSPN) (Ciardo, Muppala, and Trivedi, 1989, Marsan, Conte, and Balbo, 1984) in which transitions may fire instantaneously (when enabled) or have exponentially distributed delay between enabling and firing. Such nets can be transformed into discrete-state, continuous-time Markov processes so that analytic solution techniques may be employed to extract both steady-state and transient information.

Nevertheless, the representation capability of GSPN's is limited. In particular, token routing is difficult to effect, yet important for accurate representation of real system components such as disk cylinder request locations, process-processor bindings, and cache hit/miss behavior. Further, non-exponential timing is common in real systems, but difficult to represent. Proponents of GSPN's might argue that

any distribution with rational Laplace transform can be represented as a cascade of exponentials (Trivedi, 1983). The exception, of course, is that the poles of the transform must have negative real part, which precludes the most common non-exponential timing, namely, constant (e.g. interval timers, in-line code execution, etc.). Analytic solution procedures for Petri nets with non-exponential timing continue to impose major restrictions on net structure, e.g. at most one non-exponential transition may be enabled at any instant. Substantial effort is now being directed toward removing such restrictions (Choi, Kulkarni, and Trivedi, 1993, German and Lindemann, 1993), but the resulting stochastic processes are highly non-trivial.

The purpose of this paper is to describe an alternative approach to computer systems modeling with stochastic Petri nets. In particular, we have distilled from systems of real interest to industrial designers a minimal set of modeling capabilities that is sufficiently rich in representation to allow strong confidence in the performance predictions obtained there from. These capabilities are collected in a modeling tool (*xpetri*) that has a complete description in seven lines and yet provides the desired, expanded representation capability. Although this expanded capability pushes us outside the realm of current analytic solution techniques, we achieve favorable model solution times by compiling net descriptions into multi-threaded, customized simulation engines wherein transitions need not fire in strictly time-sequential order.

The paper is organized as follows. In section 2, we describe the *xpetri* model specification language and the representation capabilities it provides. In section 3 we describe the X-windows interface and in section 4 the multi-threaded compiler. Section 5 contains a generic procedure for workload modeling, largely due to Haring (1983), that allows us to exploit some important capabilities of *xpetri*. Section 6 contains an example, an investigation of the benefits of anticipatory seeking to reduce service time on disks. Conclusions follow in section 7.

2 XPETRI LANGUAGE SPECIFICATION

We now discuss the components of the *xpetri* language, whose complete specification is shown in figure 1. Most often, an X-windows graphical interface, described in the next section, is used to generate a model description in this *xpetri* language. However, the model description is readable ASCII text and can be created or modified with an ordinary text editor. The model description is then routed as input to either a standard, discrete event simulation engine, *xpsi*, or the compiler engine, *xpsc*. The simulation engine is intended for rapid prototyping and debugging, and has facilities for tracing all transition firings and token movement.

Model description lines beginning with '%' are comments. The graphical interface does not generate comment lines, but since the model description in the *xpetri* language can be edited directly, comments can serve as helpful guide posts. A line beginning with 'd' signals that the model specification is complete. The other five line specifications require some detail.

p A line beginning with character 'p' signifies a place specification. It is followed by a desired name for the place (a character string), and the number of initial tokens. If this number is greater than zero, the tokens must be specified individually by color. Colors are simply non-negative integers used to carry information that can control token routing, transition enabling, and transition firing time.

a A line beginning with character 'a' signifies an arc specification. Each arc specification requires the name of a place and the name of a transition. Arcs connecting places to transitions may be inhibitor arcs; in this case the attached transition is enabled only if the attached place is empty. Arcs connecting transitions to places may be probabilistic; in this case a probability must be supplied. When the attached transition fires, a token is deposited in the attached place with the specified probability.

o A line beginning with character 'o' calls for an output report on the named place or transition. Place output reports give average place utilization and average token residence time. Transition output reports give throughput, i.e., firing rate over the simulated time period.

t A line beginning with character 't' signifies a transition specification. The desired transition name is followed by specification of a firing time distribution; this specification is simply an index into a list of built-in distribution families:

- instantaneous
- constant
- uniform
- exponential
- load-dependent exponential

p	<name>	<token count>	[token list]		
t	<name>	<dstrbtn>	[prmtrs]	<group>	<corr.>
	<colorfct.>	[prmtrs]	<rstrctn count>	[place color ...]	
a	<place>	<transition>	<inhbtr|no>		
a	<transition>	<place>	<prbblstc|no>	[branch prob.]	
o	<place|trans.>				
%	comment				
d					

Figure 1: *Xpetri* Language Specification

- empirical
- linear color distance
- subnet
- user-defined

Most of these distribution choices will be illustrated in section 6, but the last two deserve comment. The "subnet" choice indicates that the firing time will be that required for a single (random) execution of the subnet named as a first parameter. This assumes that the indicated subnet requires finite time to reach an absorbing state. Subnets may be specified (recursively) to arbitrary depth. The "user-defined" choice requires a first parameter that is a file name containing a C code function of the same name. The remaining parameters are passed to the C code function, whose execution returns the firing time. In general, these distributions have a varying number of parameters, specified after the distribution choice index.

Transitions are given group names, and all transitions with the same group name have correlated firing times. The desired correlation is specified as a real number $K \in$ [-1.0, 1.0] following the group name. Sampling from the distributions is adjusted to approximate the specified correlation as follows: When transition i becomes enabled, it locates all other concurrently enabled transitions in its group. If there are none, a random number $r_i \in [0, 1]$ is used to select from the firing time distribution in the standard fashion, that is, *firing time* $= F_i^{-1}(r_i)$, where F_i is the distribution function attached to transition i. However, if there are other enabled transitions in i's group, one of these is selected at random, call it j. The last value $r_j \in [0, 1]$ used to select from j's distribution is then used in conjunction with the group correlation factor, K, to determine the new selection value $r_i \in [0, 1]$. With probability $|K|$ we let $r_i = r_j$ (let $r_i = 1 - r_j$, if $K < 0$), and with probability $1 - |K|$ we select r_i at random from $[0, 1]$.

The relationship between the selection values r_i and r_j is easily expressed. Consider the case $K \geq 0$. If R_j is a random variable with uniform(0,1) distribution, and R_i is a random variable whose dependence on R_j is as described, then the conditional density of R_i is

$$f_{R_i|R_j}(r_i|r_j) = \begin{cases} K\delta & \text{if } r_i = r_j \\ 1 - K & \text{if } r_i \neq r_j \end{cases}$$

where δ denotes unit impulse. We then have

$$\begin{aligned} E[R_i|R_j = r_j] &= Kr_j + (1 - K)/2, \\ E[R_iR_j] &= K/3 + (1 - K)/4, \\ COV[R_i, R_j] &= K/12, \text{ and} \\ \rho[R_i, R_j] &= K \end{aligned}$$

The case $K < 0$ is by like argument.

The color of the output tokens is determined, like the firing time distribution, from a list of built-in function choices:

- black
- constant color
- color from place
- color Markov from file
- color i/o from file
- color stream from file
- user-defined

Again, most of these choices will be illustrated in section 6. All but the first require parameters. For example, a common choice is to use the color found in a specific enabling input place ("color from place"), in which case the parameter is the name of the input place.

Finally, an integer number of enabling restrictions can be specified. A restriction is a place-color pair, and the transition will not enable unless a token of the specified color appears in the specified place.

Additional details regarding these features will be provided in the discussion of the example.

3 THE X-WINDOWS INTERFACE

The *xpetri* modeling tool uses a Motif-based graphical interface to allow fast specification of system models. Nets are drawn using a three button mouse. Buttons one and two are used to create and move places and transitions. Button three is used to create arcs, with optional guide posts for curved (spline) or segmented arcs, as well as to select objects for attribute editing. All places, transitions, and arcs are created with attributes copied from user-defined defaults. An object's attributes can be edited by selecting the object and opening the appropriate dialog box. The token list attached to a place can be edited from the place dialog box. Transition firing time distributions, color output functions, group correlations, and enabling restrictions can be edited using the transition dialog box. Inhibitor, probabilistic, and segmented arc attributes can be selected from the arc dialog box.

Place and transition names default to p# and t#, numbered consecutively. Screen names are limited to twenty characters to reduce screen clutter. Long names, used for the solution engine input, and the short screen names can be edited from the dialog boxes. Drawing grid and snap to grid conveniences are available. There are no restrictions on net size. An infinite drawing area is available, with zoom in/out capabilities. Nets can be output to postscript files, *xpetri* graphical files, which can be reloaded by the interface, and *xpetri* language files for routing to the solution engine(s).

It is important to note that this interface is more than a convenience: the solution engines, *xpsi* and *xpsc*, are, of course, unable to detect semantically correct input nets that contain missing or extra arcs. For large models such as that recently undertaken at DG, a complete MC88110 processor, user detection of these arc problems is extremely difficult and time consuming. This moving-window, graphical approach alleviates most such problems.

4 THE SIMULATOR COMPILER

Extensive use of Petri net models to guide large scale systems architecture and design quickly put pressure on our ability to simulate large nets for long periods of time. We have elected to address this problem in two ways. First, we have constructed a compiler for Petri nets. This compiler, *xpsc*, reads model descriptions in the *xpetri* language. Its output is a C program which can be compiled into a simulator for the described net. The compiler generates one C function for each transition. The function is tailored to handle the specific enabling function, firing time distribution, and output color selection.

The second optimization is to use posix threads, one per transition. While this adds some overhead, it allows us to reduce elapsed time when running on a multi-processor system. Each transition has an associated lock. A transition always holds its own lock except when waiting for an event. The algorithm used by each transition is then:

1. Wait until change in state of input places.
2. Check if enabled. If not, go to 1.
3. Sample the firing distribution.
4. Wait until time to fire, or change in state of input places.
5. Check if enabled. If not, go to 1.
6. Check if time to fire. If not, go to 4.
7. Release lock of self. Get locks on self and all neighbors. A neighboring transition is one that shares an input or an output place. Deadlocks are avoided by always obtaining locks in a fixed order.
8. Check if enabled. If not, unlock neighbors and go to 1. (This check is required because of the window in the previous step when lock on self is not held.)
9. Fire.
10. Release locks; signal neighbors to check for change of state; go to 2.

A count is maintained of runnable threads. When the last thread is about to go to sleep, it calls a routine that advances time to the next scheduled firing time. All transitions that are scheduled to fire at that time are then signaled, and the thread that called for all this activity goes to sleep unless it has just signaled itself.

Preliminary experiments with compiled *xpetri* nets have shown a significant speed advantage over interpreted nets. When simulating a large net (200 transitions, 206 places) the compiled version ran 15 times faster than the interpreted version on the same single processor machine.

The major flaw in this algorithm is that it reduces the number of active threads to 1 at each time step. This limits parallelism and prevents effective use of systems with very large processor counts. To avoid

this, we are experimenting with different ways of relaxing the constraint that all parts of the simulated net be time synchronized.

The simplest way to relax time constraints uses the concept of a *guaranteed fire transition* (gft). If each input place to a transition has only one exit arc, and none is an inhibitor arc, then this transition is a guaranteed fire transition. When a gft is enabled, we know with certainty that it will fire, and we know exactly when it will fire. Since we record the arrival time of each token to each place (for output measure computation), we can allow gft firing to proceed ahead of the rest of the network. Subsequent enabling checks compare token arrival times to the clock.

5 WORKLOAD MODELING

A detailed and accurate workload characterization is crucial in obtaining accurate performance predictions from models. Our previous disk modeling studies have shown that a priori benign assumptions regarding workload can have drastic effects on performance predictions. In (Geist, Reynolds, and Pittard, 1987) we found that a disk simulation model with measured cylinder access frequencies, measured service times, measured mean arrival rate, and an assumed Poisson arrival process yielded 500% error in mean waiting time predictions. A change of the arrival process (but not its mean) from Poisson to that induced by service completions reduced the error to 1%.

Dependencies in service request streams, such as successive disk or main memory addresses, can also strongly affect performance. The output token color choice, "color Markov from file", was provided to facilitate direct representation of such dependencies.

We now describe a procedure for effective use of this modeling feature. A *first-order Markov chain* is a state-based model in which next state probabilities depend only upon the current state. If next state probabilities depend upon the last 2, 3, or, more generally, n states visited, then the Markov chain is said to be of order 2, 3, or n. In this same terminology, an independent access model is an order 0 Markov chain.

We assume that a measured sequence of service requests in our workload represents a Markov chain of some order, perhaps order 0. Following Haring (1983), we can test for this order. Let $c(t) \in \{0, 1, 2, ..., N-1\}$ denote the resource accessed on the t^{th} request, where the total number of requests is T. Let

$$N_j(t) = \begin{cases} 1 & \text{if } c(t)=j \\ 0 & \text{otherwise} \end{cases}$$

$$N_{ij}(t) = \begin{cases} 1 & \text{if } c(t)=j \text{ and } c(t-1)=i \\ 0 & \text{otherwise} \end{cases}$$

We let π_i denote the steady-state probability that a request is for resource i (invariant measure), and let ρ_{ij} denote the conditional probability that the next resource requested is j, given that the last requested was i (transition matrix). Maximum likelihood estimators of π_j and ρ_{ij} are

$$\tilde{\pi}_j = \sum_{t=1}^{T} N_j(t)/T \tag{1}$$

$$\tilde{\rho}_{ij} = \sum_{t=2}^{T} N_{ij}(t) / \sum_{t=2}^{T} N_i(t-1) \tag{2}$$

We can then test the hypothesis

$$H_0 : \rho_{ij} = \pi_j \quad \forall i, j \quad \text{(Markov chain is order 0)}$$

versus the alternative

$$H_1 : \rho_{ij} \neq \pi_j \quad \forall i, j \quad \text{(Markov chain has order} \geq 1\text{)}.$$

From Haring (1983),

$$\chi_0^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=2}^{T} N_i(t-1)\ [\tilde{\rho}_{ij} - \tilde{\pi}_j]^2 \ / \ \tilde{\pi}_j$$

has χ^2 limiting distribution with $N(N-1)$ degrees of freedom. For such large degrees of freedom, we can regard $\sqrt{2\chi_0^2} - \sqrt{2N(N-1)-1}$ as a sample from a standard normal (Trivedi, 1983). Similar tests are available for H_0: order 1 versus H_1: order ≥ 2 and for higher orders.

If these tests indicate that an order 1 Markov chain is appropriate, as is often the case, the measured transition matrix (2) is installed in a file, whose name is the first parameter for the output color function, "color Markov from file". The second parameter is an input place. The color of the enabling token from the named input place is then used to select the row in the measured transition matrix. A random sample from the distribution represented by this row is the output color.

6 EXAMPLE

King (1990) advocates *anticipatory seek* as a technique for improving disk subsystem performance. During an otherwise idle period, the disk arm is moved to a position that is likely to be closer to the next request. It is easy to extend the uniform request distribution (or uniform plus hot-spot) treatment in

(King, 1990) to the general Markov case. Let X_i denote the random variable with distribution specified by row i of the Markov transition matrix for cylinders, i.e., $P(X_i = j) = \rho_{i,j}$. Given a last request to cylinder i, we would like to find the cylinder c that minimizes

$$\begin{aligned}\delta(c) &= E[|X_i - c|] \\ &= \sum_{j=1}^{N} |j - c| \rho_{i,j} \\ &= E[X_i] - c + 2\sum_{j=1}^{c-1} P(X_i \leq j)\end{aligned}$$

Necessary requirements on this minimizing c are then:

$$\begin{aligned}\delta(c+1) - \delta(c) &= -1 + 2P(X_i \leq c) > 0 \\ \delta(c-1) - \delta(c) &= 1 - 2P(X_i \leq c-1) > 0\end{aligned}$$

from which we can conclude that c is the median of the distribution. Thus during an idle period, if the last request was to cylinder i, we should seek to the cylinder that represents the median of the distribution given by row i of the matrix $(\rho_{i,j})$.

We can quickly explore the potential benefits of this policy. In figure 2 we show an *xpetri* net model for a disk with this anticipatory seek facility.

The transition *cpu* has exponential firing time with load-dependent rate: A nominal rate is multiplied by the number of tokens in place *cpu_queue* (three are shown, ten are used in the simulation) to yield the effective firing rate. The transition *Markov* uses the "color Markov from file" selection for color output. The color of the token in place *last_request* is used to select a row from the stored transition matrix, $\rho_{i,j}$. This transition matrix was obtained from measurements of real disk cylinder requests for 10 independent processes on a DG AViiON system, as described in (Geist, Suggs, and Reynolds, 1993). On firing, we record the cylinder (color) requested in the enabling place *last_request*, and thus this transition provides the "next-cylinder" selection.

The transitions *service* and *A_service* have menu-selected "linear color distance" firing times. Specifically, the colors (cylinders), c_1 and c_2, of the tokens in the two input places are used in:

$$\text{firing time} = \begin{cases} a + b + d \times |c_1 - c_2| & \text{if } c_1 \neq c_2 \\ a & \text{if } c_1 = c_2 \end{cases}$$

where a, b, and d are function parameters. Here we use $a = 8$ ms to represent rotation, $b = 4$ ms to represent startup, and $d = 0.025$ ms to represent seek time per cylinder.

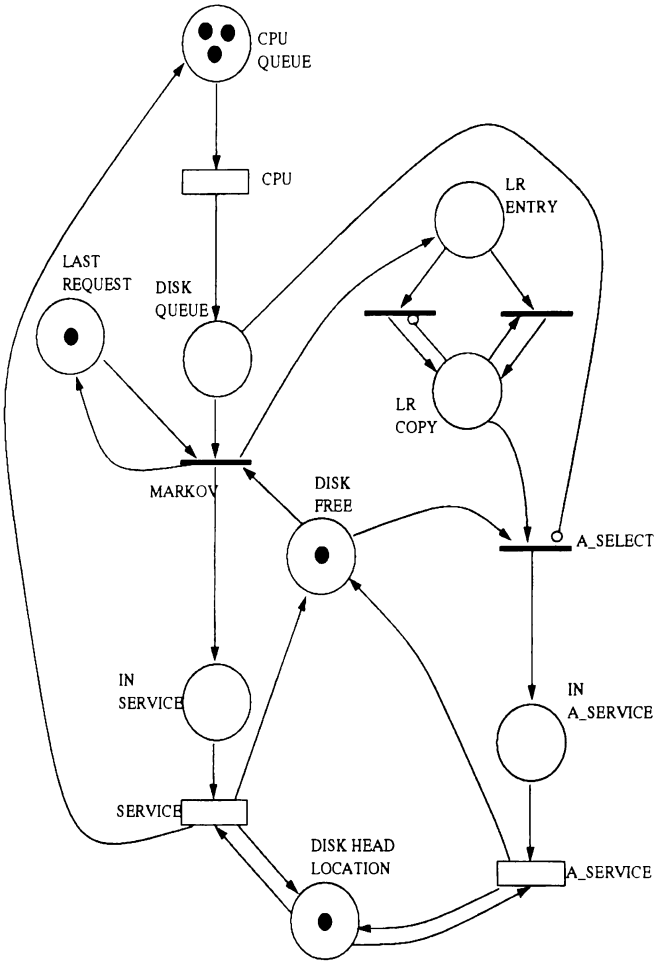

Figure 2: Anticipatory Seek Model

The *LR_entry/LR_copy* cycle simply records the last non-anticipatory seek made, and keeps it available in place *LR_copy*. Both transitions are instantaneous and select output color from the place *LR_entry*. When the disk queue is empty and *LR_copy* is not, the *A_select* transition starts the anticipatory seek; its output color is specified by the choice "color i/o from file" where the named file contains the medians of the rows of the matrix $(\rho_{i,j})$.

In table 1 we show the results of executing this net for three nominal firing rates of the *cpu* transition. In table 2 we show the results from the same net with the anticipatory seek disabled. We see that at the lowest rate the anticipatory seek was indeed beneficial to response time, and at the highest rate there is little overall effect, as might be expected, since anticipatory seeking rarely occurs. However, an interesting case is the middle rate, where we see that anticipatory seeking actually produces a detrimental effect on the predicted response time. Here, apparently, arriving requests often find themselves awaiting completion of an anticipatory seek, and the extra startup penalty is more costly than any savings in seek distance.

Table 1: Anticipatory Seek On

rate(req./ms)	0.001	0.02	0.40
service(ms)	19.29	20.02	20.12
wait(ms)	0.85	11.76	37.69
response(ms)	20.14	31.78	57.79

Table 2: Anticipatory Seek Off

rate(req./ms)	0.001	0.02	0.40
service(ms)	20.39	20.23	20.21
wait(ms)	0.69	9.01	37.83
response(ms)	21.08	29.24	58.04

7 CONCLUSIONS

We have described a new modeling tool (*xpetri*) that is based on an extension of stochastic Petri nets, and we have illustrated its use in solving computer system design problems. The tool provides both a succinct model language specification and a carefully designed, wide-ranging collection of modeling capabilities. In particular, accurate representation of workload dependencies is straightforward. In our experience, such representations are crucial to accurate performance predictions.

Solutions of models specified in the *xpetri* language are outside the realm of current analytic techniques, but a multi-threaded simulation compiler allows fast solution of large models. The X-windows interface offers speed in model specification, and, more important, improved design reliability.

In addition to the anticipatory seek results, we have used *xpetri* in sizing an on-board write cache for disk arrays, exploring alternative parity schemes for disk arrays, and predicting the performance of a copy-back cache for the MC88110.

Xpetri source code is available to the reader (via ftp) at no cost. Contact rmg@cs.clemson.edu.

REFERENCES

G. Balbo, S. Bruell, and S. Ghanta. 1988. Combining queueing networks and generalized stochastic petri nets for the solution of complex models of system behavior. *IEEE Transactions on Computers* 37:1251–1268.

H. Choi, V. Kulkarni, and K. Trivedi. 1993. Markov regenerative stochastic petri nets. In *Proceedings of the 16th International Symposium on Computer Performance Modeling, Measurement, and Evaluation (PERFORMANCE '93)*, ed. G. Iazeolla and S. Lavenberg, 339–356. International Federation for Information Processing, Rome, Italy.

G. Ciardo, J. Muppala, and K. Trivedi. 1989. SPNP: Stochastic Petri net package. In *Proceedings of the International Conference on Petri Nets and Performance Models*, 142–150. Institute of Electrical and Electronics Engineers, Kyoto, Japan.

J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. 1984. Extended stochastic petri nets: Applications and analysis. In *Proceedings of the 10th International Symposium on Computer Performance Modeling, Measurement, and Evaluation (PERFORMANCE '84)*, ed. E. Gelenbe, 507–520. International Federation for Information Processing, Paris, France.

R. Geist, R. Reynolds, and E. Pittard. 1987. Disk scheduling in System V. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 59–68. Association for Computing Machinery, Banff, Alberta.

R. Geist, D. Suggs, and R. Reynolds. 1993. Minimizing mean seek distance in mirrored disk systems by cylinder remapping. In *Proceedings of the 16th International Symposium on Computer Performance Modeling, Measurement, and Evaluation (PERFORMANCE '93)*, ed. G. Iazeolla and S. Lavenberg, 91–108. International Federation for Information Processing, Rome, Italy.

R. German and C. Lindemann. 1993. Analysis of stochastic petri nets by the method of supplementary variables. In *Proceedings of the 16th Interna-*

tional Symposium on Computer Performance Modeling, Measurement, and Evaluation (PERFORMANCE '93), ed. G. Iazeolla and S. Lavenberg, 320–338. International Federation for Information Processing, Rome, Italy.

G. Haring. 1983. On stochastic models of interactive workloads. In *Proceedings of the 9th International Symposium on Computer Performance Modeling, Measurement, and Evaluation (PERFORMANCE '83)*, ed. A. Agrawala and S. Tripathi, 133–152. International Federation for Information Processing, College Park, Maryland.

M. A. Holliday and M. K. Vernon. 1987. A generalized timed Petri net model for performance analysis. *IEEE Transactions on Software Engineering* SE-13:1297–1310.

K. Jensen. 1987. Colored Petri nets. *Lecture Notes in Computer Science* 254:248–299. Berlin: Springer-Verlag.

Richard King. 1990. Disk arm movement in anticipation of future requests. *ACM Transactions on Computer Systems* 8(3):214–229.

M.K. Malloy. 1982. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers* C-31:913-917.

M. Marsan, G. Conte, and G. Balbo. 1984. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems* 2:93–122.

Y-B. Shieh, D. Ghosal, and S. Tripathi. 1989. Modeling of fault-tolerant techniques in hierarchical systems. In *Proceedings of the* 19^{th} *International Symposium on Fault-tolerant Computing (FTCS-19)*, 167–174. Institute of Electrical and Electronics Engineers, Chicago, Illinois.

K.S. Trivedi. 1983. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications.* Englewood Cliffs, NJ: Prentice-Hall.

T. Yoneda, K. Nakade, and Y. Tohma. 1989. A fast timing verification method based on the independence of units. In *Proceedings of the* 19^{th} *International Symposium on Fault-tolerant Computing (FTCS-19)*, 134–141. Institute of Electrical and Electronics Engineers, Chicago, Illinois.

AUTHOR BIOGRAPHIES

ROBERT GEIST is a Professor in the Department of Computer Science at Clemson University. He received a B.A. degree in mathematics and an M.A. degree in computer science from Duke University in 1970 and 1980 respectively, and he received M.S. and Ph.D. degrees in mathematics from the University of Notre Dame in 1973 and 1974 respectively. His research interests are in performance and reliability modeling of computer and communication systems and stochastic modeling in computer graphics.

DARREN CRANE is virtual reality system administrator for Clemson University. He received B.S. and M.S. degrees in computer science from Clemson University in 1991 and 1994 respectively. His research interests are in performance modeling, computer graphics, and stereo lithography.

STEPHEN W. DANIEL is a Staff Specialist in the Database Engineering department of Data General's Research Triangle Park North Carolina laboratory. He received a B.S. degree in physics from Williams College in 1979 and an M.S. degree in computer science from Duke University in 1981. His research interest is in the use of performance modeling of large scale systems to improve database performance.

DARRELL SUGGS is a Senior Software Engineer for Data General Corp., RTP, NC. He received a B.S. degree in computer science from Appalachian State University in 1988, and he received M.S. and Ph.D. degrees in computer science from Clemson University in 1990 and 1993 respectively. His research interests are focused on multiprocessor workload characterization, systems modeling for advanced architectures, and stochastic modeling.