Techniques and Tools for Reliability and Performance Evaluation: Problems and Perspectives

Kishor S. Trivedi[1]* and Boudewijn R. Haverkort[2] and Andy Rindos[3] and Varsha Mainkar[4]

[1] Dept. of Electrical Engineering, Duke University, Durham, USA
[2] Department of Computer Science, University of Twente, The Netherlands
[3] Networking Systems Hardware, IBM Corp., RTP, USA
[4] Department of Computer Science, Duke University, Durham, USA

Abstract. Modelling techniques and tools of the future must meet the challenges presented by today's highly demanding and schedule-oriented developing environment. With the emergence of high performance and reliability systems the problem of how to analyze such systems has become increasingly more difficult. Traditional assumptions of independent events, exponential distributions and other such "convenient" assumptions no longer model systems realistically. Nevertheless, the demand for answering performance and reliability related questions during the design process has increased. In this paper we discuss some of the issues involved in integrating modeling and design during a product development process. We present a broad range of existing techniques of systems analysis. We also describe a variety of tools that have been developed to make the analysis process simpler.

1 Introduction

One of the greatest challenges facing today's performance, reliability and performability tool designers is to meet the rapidly changing needs of computer and communications product designers and capacity planners. These designers and planners must develop those systems that will satisfy ever more demanding customer expectations of reliability and performance. These challenges should invigorate the area of systems analysis research by introducing a vast array of real problems that stress existing theoretical techniques.

By examining these real problems commonly encountered by such designers, we should gain an insight into the capabilities an ideal analysis tool should provide. Likewise, by examining the pressures experienced by designers as they develop a product, as well as designer-analyst interactions, we should gain insight into additional features (e.g., ease of use, reusability of submodels, etc.) that a successful tool should offer. As a result of these examinations, we should be able to suggest successful ways in which such tools, as well as the systems analysis

* This research was sponsored by IBM under the IBM/Duke University Research Agreement # RAL-R93010-00.

experts that develop them, can be effectively integrated into the design process. This should lead to an increased viability of such tools, as well as a mutually beneficial relationship between tool developers and tool users.

1.1 How should realistic systems be analyzed?

A large number of real system characteristics cannot be easily modeled using present analytic techniques. Most realistic systems cannot be separated into small independent subsystems. Their detailed state representation then may be excessively large. Further, events in real systems are not "memoryless" and event times may differ by several orders of magnitude. All this implies that simple analytic models such as product form queueing networks may not be powerful enough, and detailed analytic models (such as Markov chains) may be prohibitively large and stiff. This unfortunately has led to an over-reliance by designers on discrete event simulation, which is often improperly used, or applied to situations where a simpler analytic model would suffice. When extremely accurate predictions of final product performance/reliability (especially distributions) are needed, discrete event simulation is probably the best method. However, detailed simulation models require a great deal of time to construct, parameterize, validate and solve. As an example, a recently developed detailed simulation model of an actual ATM adapter required 12 hours of machine time on a very powerful server (receiving 100% of the CPU cycles) to achieve acceptable 90% confidence intervals for the desired response time measures. The bottleneck resource in the model was utilized at only 40% of its capacity for the run. Runs with higher utilizations would have required unacceptable amounts of machine time, since run time for the same level of confidence increases exponentially with utilization.

However, when we examine the evolution of the design process of an actual product, it becomes apparent that simulation is often unnecessary. At the earliest stages of this process, a very simple, high-level analytic model of the product design is probably sufficient. Simulation would not be recommended at this stage, since actual values for many system parameters, as well as many of the actual design details, would not be available. However, comparisons of different high-level design alternatives could be made with analytic models so that the best high-level design could be chosen at the outset. As the design matures and more details of the technology, architecture, etc. are defined, more accurate analytic techniques, hierarchical approximation, and finally, detailed discrete-event simulation, might be sequentially used to analyze the system.

Using such an evolutionary modeling approach, a nearly optimal design can be developed from the beginning, avoiding costly 'improvements' made when the design is less flexible (especially when the implementation has begun). Redesign work late in the development process may also lead to delayed release of the product. This points to the value of modeling the system early on and throughout development. Such models typically alert the designer to potentially detrimental design flaws, hopefully, at a time when they can be easily fixed.

1.2 The needs of today's product developers

Today's product designers need to get out the best possible product in the shortest period of time using the least amount of resources. This driving force pulls the performance and reliability of the resultant product in two opposite directions. On the one hand, in today's very competitive market, the developer who is first to market may capture the largest share of the market. Therefore, schedules for product release are very aggressive and sacrosanct. This means that product developers cannot afford a lot of time to develop and analyze models of their product's performance and reliability. On the other hand, customers are requiring far greater reliability and performance which would suggest a need for more careful (and, possibly, more time consuming) design and analysis.

Tools that require very little time to learn to use and allow models to be constructed very quickly will therefore be highly sought after. This suggests that easy-to-use graphical user interfaces and clear, self-explanatory menus should be developed for all future tools that target an industrial user-base. Many performance/reliability tools on the market already provide some of these features. Furthermore, the ability to define reusable submodels of system and network components that can be combined to produce large models and that provide the capability of multiple instances in a given model are an absolute must. A number of tools are avoided in industry because of their inability to provide these capabilities.

Another important aspect in model specification is the ability to specify models in a hierarchical way since this allows one to stay close to the generally hierarchically designed systems. When supported in a flexible way, this also allows for the building of a database (a model-base) of predefined model building blocks. Preferably, the hierarchical model structure should be exploited in the model solution phase, however, this is not always easily done and it almost always introduces approximation error.

Tools that provide alternative analytic methods that require less time and memory to generate and solve the models, in conjunction with simulation, are extremely desirable. Such tools with multiple analysis capabilities would fit very well with the evolution of the design process of an actual product as described before. The conjunction of several techniques in a single tool might allow the user to input the appropriate details of his product only once, and then analyze it using different methods, depending upon his needs for accuracy vs. speed of execution. It would also facilitate the use of hybrid techniques and hierarchical model development.

Ideally a tool should also have an integrated method of calibration for its models. That is, it should be closely coupled with databases maintained by manufactures about their products. These databases could provide information for model calibration such as failure rates, device speeds, repair times and so on. Using the measures contained within these databases, it should not only be easy to parametrize a single run, but also a series of experiments with changing model parameters allowing for parametric studies. Tools with such multiple capabilities could therefore satisfy the pull by today's market forces to develop

models quickly and in a timely manner so that products are brought to market in the shortest period of time.

1.3 Relationship Between Designers and Performance Experts

In the past, especially in large corporations, one or more performance departments have existed independently of those departments responsible for the actual design and implementation of products. The justification for such an arrangement has typically been that such separate departments serve as centers of competency, whereby the members can consult with one another when difficult problems arise in their analyses. Such an arrangement also fosters departmental experts who can provide focus specifically on techniques that will make the work of theoretically analyzing products more accurate and efficient.

However, some inherent problems exist in this arrangement. Because analysis and design departments are separate, conflict often arises when the analyst exposes serious performance/reliability flaws in the product design. Although sometimes the feedback is appreciated, it more often leads to a sudden lack of cooperation between the two groups. The design team may suddenly make it very difficult for the performance team to obtain additional design details. This points to another drawback of the arrangement of separate departments. The analyst typically requires time from the designer's often busy schedule to verify his understanding of the design. As an outsider, it is a very difficult task for the analyst to obtain an accurate picture of the details of the design from reams of poorly indexed design documents that are constantly changing as the product design evolves. It is especially difficult to keep up with such changes in both the hardware and the software (some knowledge of both are needed for such models).

The availability of time from a product developer's schedule has become increasingly more difficult to obtain in today's schedule-driven, "more efficient" (a euphemism for over-worked) working environment. Given the cost to a designer of taking the time to cooperate, one might be sympathetic to their reluctance to assist the analyst. If the analyst simply verifies the correctness of the design (that is, it meets performance specifications, etc.), his work is often viewed as superfluous; if he exposes problems, his work is viewed with hostility. Therefore, an analyst should also provide insight into or solutions to problems encountered during the analyses, as well as improvements to the existing design, if his work is to be properly appreciated. Furthermore, those improvements should be available in a timely manner during the design cycle so that they can be implemented.

This has led to several strategies to reduce the problems inherent in a separate department arrangement. For those organizations that were large enough to allow a separate high-level design department for a product, this department was often combined with the analysis department. However, with a movement toward a smaller, more efficient workforce, what is more typically encountered is a small single group with a very broad range of responsibilities for the design and implementation of a single product. In an extreme reaction to needed cuts in work force, some of the groups have no one officially responsible for analyzing the performance/reliability of their product. They therefore operate at the risk

of encountering major design problems surfacing late in the implementation or prototype phase.

Because of such potential exposure, these tighter design groups often allocate one person to be responsible either part-time or full-time to these performance-related problems. Since the person is a part of the design and implementation team, he is able to track the design from the very beginning and is not likely to be 'frozen-out' when performance-related problems in the design are encountered, since he is expected to provide some of the solutions. However, one major problem is the availability of enough qualified performance analysts that can be distributed among various product design groups. This inherent variability in talent can be smoothed out either by means of improved training or by means of better tools.

1.4 The focus of this paper

Recently, there have appeared a number of papers with surveys similar to this one. Most notably, we mention Meyer [54] who overviews the history of the concept of performability after he introduced it in the early eighties [51, 52]. De Souza e Silva and Gail [74] discuss the specific technique known as uniformization, which is now known to be the method of choice for transient analysis of continuous-time Markov chains (CTMCs). Trivedi *et al.* [79] present mathematical evaluation techniques for performability. Haverkort and Trivedi [32] overview specification techniques for Markov reward models. Mulazzanni and Trivedi [59] overview tools for dependability, as do Geist and Trivedi [24] and Johnson and Malek [38]. A description of dependability analysis of real-time systems can be found in [78].

This paper differs from the above ones in that it addresses reliability, performance and performability evaluation tools and techniques from the user's perspective. Former papers often were restricted to either performability, or to just tools or to just techniques. In this paper you will find a mix of it all.

The organization of the paper is as follows. In Section 2 we try to answer questions of the form "What are we interested in?" and "Why do we actually model?". Answers to these questions give us directives to approaches towards reliability and performance evaluation, which are discussed in Section 3; this section still focuses on techniques. Tools supporting the various categories of techniques are then discussed in Section 4. Problem areas and future perspectives are discussed in Section 5. The paper is concluded in Section 6.

2 What are we interested in?

In this section we try to answer the question "Why are we doing all this?" For simplicity, we split the question in a number of subquestions. In Section 2.1 we discuss system aspects we are interested in, for various classes of systems. Then, in Section 2.2, we discuss measures that quantify the interest expressed earlier. Various approaches towards how to derive these measures are discussed in Section 2.3.

2.1 System aspects of interest

A first and important distinction in answering this question, is whether we take the viewpoint of a system user or of a system provider. This distinction is most clear when thinking of the system as a public data network as provided by many telephone companies. System users want a good *quality of service* (QoS), whatever that may be, whereas the provider wants a high profit. The latter implies that the system should do what the user wants it to do, but at the lowest possible cost.

The QoS asked for by a user is often subjective, e.g., a user wants a good video channel quality, if he is intending to use the network for video transmission. How this good quality is expressed in terms of bit rates, bit error rates or switch blocking probabilities is not easy determined. These latter measures express what is known as the *objective quality of service*. For example, a user of a parallel processing system will most likely be interested in the throughput and turn-around time of his jobs, not so much in the degree of parallelism achieved by the system. The latter is of interest to the system designer or for the cooperation "selling" parallel processing capacity.

The above distinction is important and should be kept in mind when doing practical evaluation studies; the type of viewpoint has implications on the required results.

2.2 Measures of interest

The distinction made in the previous section directly has its implications for the measures we want to evaluate. We distinguish between *task-oriented* measures and *system-oriented* measures. Task-oriented measures typically say something about the end-to-end performance as perceived by system users. Examples are the end-to-end throughput or delay, or the expected performance level over some time interval of system usage. System-oriented measures say something about how, internally, the system performs its tasks. Examples are the average queue length, the number of operational components at some time, or the utilization of a server. As such, these measures are not so much of interest to the system user, although they are intimately related with the task-oriented measures.

With pure performance evaluation, both task- and system-oriented measures can be obtained. Examples of the former are job response or waiting times, examples of the latter are average number of occupied buffers or utilizations. These measures suffice if the assumption that the system never fails is acceptable.

With pure dependability evaluation, the emphasis is on deriving system-oriented measures, although safety measures are user-oriented as well. However, we think that system users are not so much interested in high availability as in a high probability that the task they want to be performed are actually performed. For that reason, performability measures seem more suitable.

Performability evaluation mainly aims at providing user-oriented measures. It has been claimed by other authors as well, that the evaluation of performability comes closest to the evaluation of objective QoS.

Another distinction that one often encounters is whether the measures are derived over a time interval or for some particular time-instance. Interval-of-time measures include the steady-state measures, when the interval is taken to be infinitely long. Also, cumulative measures over finite time horizons (see Section 3) belong to this category. Instant-of-time measures are also called transient measures; they express the performability (or performance, or reliability) of a system at some time point t.

2.3 Methods of evaluation

We distinguish three classes of methods for system performance, dependability or performability evaluation: measurement-based, model-based and hybrid methods.

Measurement-based evaluation (also called empirical evaluation) requires one to have at one's disposal a measurable system. Apart from the fact that this is often not the case in the design phase of the system, performing measurements is often expensive since it requires special purpose hardware and software. Also, for dependability modeling purposes, measurement is difficult. Dependability events, i.e., system failures, do not occur that often in highly-reliable systems, requiring extremely long measurement sessions.

Measurement studies are often done to determine system parameters that are later to be used in a modeling study, or to validate a model.

As an alternative to measurement-based evaluation, a model-based evaluation can be used. A system model can be very simple, e.g., some mathematical formula relating the system performance to the system parameters, or very complex, e.g., a large system of differential equations or a complex simulation program.

Once a model has been constructed, it needs to be solved. This can be done using discrete-event simulation or using analytical techniques. Analytical techniques can be fully symbolic, semi-symbolic or numerical. Fully symbolic analytical techniques provide simple functional relations between system parameters and the measure of interest ($E[N] = \rho/(1-\rho)$ in the M/M/1 queue). Semi-symbolic analytical techniques provide mathematical relations between system parameters and measures of interest, however, some parameters in the relation are to be determined by numerical technique (ACE [50]). Lastly, some analytical techniques require numerical solution such as linear-system solution or the solution of a differential equation.

Another distinction that sometimes is made is whether the solution of the model requires the whole model state space to be explicitly generated or not. The most well-known example of the former is the solution of a large but finite Markov model. An example of the latter is the use of an MVA solution procedure [65] in a product-form queueing network or the use of fault-trees for reliability analysis [23].

Most of the useful evaluations in practice use a judicious combination of different modeling approaches with measurements. For example, fault-injection simulation (or actual measurements on a prototype) can provide coverage-like

parameters in an analytic reliability model. A performability model of a multiprocessor system may have a Markov reliability submodel and a product-form queueing network as a performance submodel.

3 Approaches to performance and reliability evaluation

In this section we discuss four approaches to performance, reliability and performability analysis. In Section 3.1 we discuss so-called *non-state space* methods, thereby meaning that explicit knowledge and enumeration of the state space of the model is not needed for evaluation purposes. In Section 3.2 we discuss Markov chain based methods, and in Section 3.3 stochastic Petri net based models. In Section 3.4 we discuss hierarchical and approximate modeling approaches.

3.1 Non-state space

With this class of models, our aim is to compute required performance and reliability measures without explicitly generating overall state space. This is a very nice property as state-space sizes tend to increase exponentially with the problem size. Three of the most well-known non-state space methods are *product form queueing networks* (PFQNs), *fault trees* (FTs) and *matrix geometric methods* (MGMs).

With PFQNs one needs to specify a number of resources (the queues and servers) as well as the way in which customers make use of these resources. The queues form the active elements that can serve customers in an order governed by one of the scheduling discipline: FCFS, LCFSPR, PS, or IS. The customers travel through the QN according to routing chains. Customers may be grouped in classes. At every queue, customers belonging to a specific class request a general differential service time distribution (at FCFS stations only exponential service time distributions are allowed). After service completion, the customer proceeds to the next queue along its routing chain. The state of a PFQN model is a vector consisting of the number of customers of each class residing at each queue. The completion of a service at a particular queue or the arrival of new jobs causes a state change. Instead of solving such a model at the state space level, one can employ special techniques that exploit the specific model structure and that are much less computation and memory intensive. The convolution approach introduced by Buzen [9] and the mean-value analysis (MVA) introduced by Reiser and Lavenberg [65] and their derivatives constitute the common techniques. With the former an efficient recursive scheme is used to calculate normalizing constants which can be used in straightforward calculations for derived performance measures such as average queue length, utilizations and throughputs. In the MVA approach, a recursive scheme in terms of the average performance measures is developed.

Fault-trees(FT) are a commonly used non-state space (also called combinatorial) method for reliability (availability, safety) analysis. With FTs the conditions under which a system fails, are expressed as a tree structure containing

logic gates. Component failure events form the leaves of the tree. Subsystems and components must have stochastically independent failure behavior.

The measures of interest are normally computed using combinatorial methods, that is, the system failure event is expressed as a logical function of the failure events of subsystems and components. Dependability measures (such as reliability or safety at time t or mean time to system failure) of interest are then computed numerically or symbolically directly from the tree. Algorithms for fault-tree analysis can be found in [55].

Specification techniques that are very closely related to FTs are reliability block diagrams and reliability graphs. For further details, the reader is referred to [55] and [66].

With MGMs, the repetitive structure of the underlying Markov chain in many queueing models is exploited. When observing the (embedded) generator matrix $\mathbf{Q}$ of many queueing models, it appears that apart from a number of so-called boundary columns, from some point onwards, all columns are the same, except for the fact that they "shift down". This appears most notably in the M/M/1 queue, however, also in more complex queueing systems this structure can be observed. In the latter case, the columns are often columns of matrices rather than scalars. Due to this special structure, the steady-state probabilities can be grouped in so-called *levels* and the steady state probability vector for level i, i.e., $\underline{z}_i$ can be expressed as

$$\underline{z}_i = \underline{z}_0 \mathbf{R}^i, \quad i = 0, 1, \cdots \tag{1}$$

i.e., the steady state probability vectors per level exhibit a geometric solution in terms of the matrix $\mathbf{R}$. The basis of the recursive solution is obtained by solving a system of linear equations corresponding to the repeating portion of the global balance equations and the normalization equation. The matrix $\mathbf{R}$ follows from a quadratic equation that can easily be solved iteratively. The size of $\mathbf{R}$ equals the number of states per level, typically small and finite. By contrast, the original Markov model solution would have required an infinite system of linear equations.

3.2 Markov reward models

In this section, we present a unified framework for performance, reliability and performability models in terms of Markov reward models. A comprehensive account of Markov reward models for performability analysis appears in [79]. Several references on solution methods for the measures defined below can be found in [15, 29, 64, 74, 79].

Definitions

Let $\{\Theta(t), t \geq 0\}$ be a continuous-time finite-state homogeneous Markov chain (CTMC) with state space Ψ. A constant reward rate r_i is associated with each state i of the Markov chain. With the reward rate specifications, the CTMC can

be termed as *Markov reward model* (MRM). If the MRM spends τ_i time units in state i, then $r_i\tau_i$ is the reward accumulated. It is also possible to associate reward rates with the transitions of the CTMC. For more basic information on MRMs, refer to [37].

Let $\mathbf{Q}$ be the generator matrix and $\underline{P}(t)$ be the state probability vector of the MRM. Here $P_i(t)$ denotes the probability of the MRM being in state i at time t. The transient behavior of this MRM is given by the Kolmogorov differential equation:

$$\frac{d\underline{P}(t)}{dt} = \underline{P}(t)\mathbf{Q} \quad , \tag{2}$$

given the initial state probability vector $\underline{P}(0)$. The steady-state probability vector $\underline{\pi}$, assuming that it exists and is unique, is obtained by setting the l.h.s. in Equation 2 to zero:

$$\underline{\pi}\mathbf{Q} = 0 \quad , \tag{3}$$

subject to the condition $\sum_{i\in\Psi} \pi_i = 1$. Here π_i is the steady-state probability of the MRM being in state i. Let us now define a cumulative state vector of the MRM as $\underline{L}(t) = \int_0^t \underline{P}(x)dx$. $L_i(t)$ denotes the expected total time spent by the MRM in state i during the interval $[0, t)$. Integrating Equation 2, we obtain:

$$\frac{d\underline{L}(t)}{dt} = \underline{L}(t)\mathbf{Q} + \underline{P}(0) \quad . \tag{4}$$

For MRMs with absorbing states, the state space Ψ can be partitioned into two subsets: Ψ_A (absorbing states) and Ψ_T (transient states). Corresponding to the non-absorbing states, the submatrix $\mathbf{Q}_T$ of $\mathbf{Q}$ can be defined. The mean time spent by the MRM in state i is given by $\tau_i = \int_0^\infty P_i(x)dx$, which can be computed by integrating Equation 2 from 0 to ∞:

$$\underline{\tau}\mathbf{Q}_T + \underline{P}_T(0) = 0 \quad . \tag{5}$$

The mean time to absorption in such a Markov chain is given by:

$$MTTA = \sum_{i\in\Psi_T} \tau_i \quad . \tag{6}$$

Performability Measures

Let $\Upsilon(t) = r_{\Theta(t)}$ be the instantaneous reward rate of the MRM. The accumulated reward over a period of time $[0, t)$ is given by:

$$\Phi(t) = \int_0^t \Upsilon(x)dx = \int_0^t r_{\Theta(x)}dx \quad . \tag{7}$$

The expected instantaneous reward rate at time t of the MRM is:

$$E[\Upsilon(t)] = \sum_{i\in\Psi} r_i P_i(t) \quad . \tag{8}$$

The expected reward rate in steady-state of the MRM is:

$$E[\Upsilon_{ss}] = \sum_{i \in \Psi} r_i \pi_i \quad . \tag{9}$$

The expected accumulated reward in the interval $[0, t)$ of the MRM is:

$$E[\Phi(t)] = \sum_{i \in \Psi} r_i L_i(t) \quad . \tag{10}$$

The expected time-averaged reward in the interval $[0, t)$ is given by $\sum_i r_i L_i(t)/t$. For an MRM with absorbing states, expected accumulated reward until absorption is:

$$E[\Phi(\infty)] = \sum_{i \in \Psi_T} r_i \tau_i \quad . \tag{11}$$

The distribution of $\Upsilon(t)$ is computed as:

$$P[\Upsilon(t) \leq x] = \sum_{r_i \leq x, i \in \Psi} P_i(t) \quad . \tag{12}$$

The distribution of accumulated reward until absorption and distribution of accumulated reward over a finite period of time can also be computed.

Let the time to accumulate a given reward r be denoted by $\Gamma(r)$. Then the distribution of $\Gamma(r)$ is known once the distribution of accumulated reward is known [40]:

$$P[\Gamma(r) \leq t] = 1 - P[\Phi(t) < r] \quad . \tag{13}$$

For example, the distribution of time to complete a job that requires r units of processing time on a system which is modeled by an MRM can be computed in this manner.

Dependability Measures

In a dependability model, a reward rate of 1 is assigned to all the system operational states and reward rate 0 is assigned to all the system failure states. The instantaneous availability of the system is then $E[\Upsilon(t)]$ and steady-state availability is $E[\Upsilon_{ss}]$. The cumulative operational time of the system in time interval $[0, t)$ is $E[\Phi(t)]$. Interval availability is the proportion of time a system is operational in a given interval of time and it is given by $E[\Phi(t)]/t$. Measures related to time to first system failure are also of interest. To compute these measures, all the failure states are made absorbing (outgoing arcs from these states are removed). Reliability is then given by $E[\Upsilon(t)]$. The lifetime (analogous to cumulative operational time) [74] of the system in interval $[0, t)$ is $E[\Phi(t)]$ and mean time to system failure (MTTF) is $E[\Phi(\infty)]$. The repairability of the system is computed by making all the operational states absorbing and reversing the reward rates (i.e., making reward rate 1 to 0 and vice-versa) and computing $E[\Upsilon(t)]$.

Performance Measures

In a performance model, queue length at a resource may be the reward assignment to a state. Then $E[Y_{ss}]$ and $E[Y(t)]$ will yield the average steady state and average transient queue length, respectively. In a like manner, throughput, buffer overflow probability etc. can be obtained as reward measures.

In a performability model, reward assignment is typically computed from a performance model (throughput, probability of violating a response time deadline) which is evaluated for different states of a failure/repair model. Throughput and response time deadline violation probability can then be computed including the effects of failure/repair.

3.3 Stochastic Petri net models

Stochastic Petri nets (SPNs) have been developed as extensions to the non-timed Petri nets by Molloy [56], Ajmone Marsan *et al.* [2] and Meyer *et al.* [53]. Although at first primarily used for the performance analysis of computer systems, SPNs are increasingly being used in other application areas such as performability and dependability evaluation.

When using an SPN specification technique, one has to define a set of places P, a set of transitions T, and a set A of arcs between transitions and places or *vice versa*: $A \subseteq (P \times T) \cup (T \times P)$. Each place can contain zero or more tokens. Graphically, places are depicted as circles, transitions as bars, tokens as dots (or integers) inside circles, and arcs as arrows.

The distribution of tokens over the places is called a marking and corresponds to the notion of state in a Markov chain. All places from which arcs go to a particular transition are called the input places of that transition. All places to which arcs go from a particular transition are called the output places of the transition. A transition is said to be enabled when all of its input places contain at least one token. If a transition is enabled it may fire. Upon firing, a transition removes one token from all of its input places and puts one token in all of its output places, possibly causing a change of marking, i.e., a change of state.

The firing of transitions is assumed to take an exponentially distributed time. Given the initial marking of an SPN, all the markings as well as the transition rates can be derived, under the condition that the number of tokens in every place is bounded. Thus a finite Markov chain is obtained.

The reward rates are described as a function of the markings, i.e., at the SPN level. The reward rates and the Markov chain together yield a MRM [16].

Various extensions have been made to the basic SPN model described above [2, 16, 53]. These include arcs with multiplicity, a shorthand notation for multiple arcs between a place-transition pair, immediate transitions that take no time at all to fire (depicted as thin bars), and inhibitor arcs from places to transitions that prevent the transition to fire as long as there are tokens in the place (depicted as lines with a small circle as head). Also, more flexible firing rules have been proposed, most notably the introduction of *gates* in stochastic activity

networks (SANs) [53, 68] and guards or enabling functions in Stochastic Reward Nets (SRNs) [16].

Normally, SPN (look-alike) models are solved via an underlying MRM which can automatically be derived, thereby using the wide variety of available techniques as indicated in the previous section. For very large models when statspace generation is prohibitive, simulation can be used as well. Especially in the field of dependability and performability evaluation there might be a need for the incorporation of fast-simulation techniques such as importance sampling [26] or injection simulation [57]. For a restricted class of SPN models product form solutions are available, see e.g., [36]. Given such a structure, MVA [19] and convolution [18] schemes have recently been devised.

3.4 Hierarchical and Approximate Models

With hybrid approaches, two or more techniques are combined in the construction and solution of a single model. Very often this takes the form of hierarchical modeling. Submodels are specified in one formalism and the result of the submodel analysis are embedded in a higher-level model. Hierarchical modeling, however, is not always hybrid modeling. The decomposition result in PFQN is a form of non-hybrid, hierarchical modeling. Beginnings of a theory of hierarchical models of SPN type can be found in [17]. There is less general theory for hybrid modeling. We therefore mention some published approaches.

For pure performance studies, Balbo *et al.* combined queueing networks and GSPNs for the analysis of system models with non-product form characteristics [3]. The non-product form parts of the model are solved using GSPNs, the results of which are used in load-dependent queueing stations that fall in the category of PFQNs.

With the software tool SHARPE (see Section 4) many model types can be analyzed, using a variety of techniques. The result of one analysis can be embedded in other models. This can be done in a cyclic way as well; in that case fixed-point iteration techniques are needed to solve the overall model.

In the dynamic queueing network concept proposed by Haverkort *et al.* [28, 29] queueing networks are used for describing performance aspects, and GSPNs are used to describe dependability aspects of fault-tolerant computer systems. An overall model is not explicitly constructed, instead, an approximate solution based on behavioral decomposition as is common in performability evaluation is utilized.

For non-product-form networks (NPFQN), a number of approximate techniques exist. A major concern with these techniques is in the characterization of their error under a wide variety of realistic network situations. A number of methods exist for closed non-product-form networks, including Marie's algorithm [49]. A wide variety of methods also exist for general open networks. A popular method, on which a number of tools are based (e.g., QNA [81]) is a two-moment decomposition method that was developed by Whitt to handle networks with general independent interarrival and service time distributions.

Mean waiting times are predicted using a two-moments of service and interarrival times. Mean interarrival times are computed by solving the standard traffic equations. The coefficient of variation of the interdeparture time is computed by means of Marshal's formula, using an approximation for the mean waiting time. As the use of Marshal's formula implies, all interarrival processes to resources within the network are assumed to be renewal processes. Under this assumption, using heuristic methods developed by Albin and Whitt [81] that are based on large amount of empirical evidence, a linear system of simultaneous equations is derived to solve for the coefficient of variation of interarrival times to all resources.

4 Tools for performance, reliability and performability evaluation

For the basic approaches distinguished in Section 3, we discuss a number of software tools that support them. Due to space limitations we can not go into much detail, however, we provide references that can be tracked down for further study.

4.1 Non-state space

In the reliability domain, the tools SHARPE [66] and HARP [22] can both solve fault-tree models. SHARPE also solves reliability block diagrams and reliability graphs. SHARPE can provide semi-symbolic expressions (in terms of t) for the reliability function.

Performance modeling packages QNAP [63] , RESQ [47], and HIT [5] support a wide variety of PFQN analyses, such as MVA and convolution algorithms. The tool HIT also supports nice facilities for hierarchical modeling, both exact (Norton's theorem, and approximate). SHARPE also solves multiclass PFQN models using the MVA algorithm and series-parallel task precedence graphs.

Two tools that make use of MGMs are MAGIC developed by Squillante [76] and Xmgmtool developed by Haverkort [34] respectively. MAGIC allows one users to input the regular (repeating) block-structure of the Markov generator matrix. It subsequently calculates the matrix $\mathbf{R}$ and the initial vector of the recursion $\underline{z}_0$. Xmgmtool provides similar facilities, however, it also provides facilities to specify queueing systems in terms of their interarrival and service time distributions (both of phase-type). The underlying regular matrix structure is subsequently generated and solved. The output is also given in terms of the queueing systems originally specified.

4.2 Markov reward models

With SHARPE [67], MRMs can be input at the state level by a simple enumeration of the state-change rates and the reward rates per state. SHARPE then solves Markov and semi-Markov reward models for their steady-state, transient

and cumulative behavior. Specification is textual, but abilities include solving the model for many different parameters using "loop" specifications.

The textual tool MARCA has been developed by Stewart [77] at North Carolina State University. Although not really an SPN tool, its modeling constructs, i.e., buckets, balls, and transitions, can easily be interpreted in an SPN context as places, tokens and transitions. Emphasis in the the tool is on advanced steady state numerical solvers.

There are many tools available that solve models via the underlying MRM. The largest class of such tools exists in the context of SPNs; that is why we discuss them separately. However, some other tools based on other modeling paradigms are discussed below. In particular we address queueing network (QN) based tools and tools based on production rule systems (PRS).

The tool QNAP2, developed at INRIA by Potier *et al.* [63, 80], is a general QN-based performance analysis tool which supports simulation, (approximate) product-form solutions as well as a numerical solution based on an underlying MRM. In fact, given a textual representation of a QN, the QNAP2 model is transformed to an intermediate model similar to the MARCA model . Only steady-state measures are computed.

The performance analysis tool NUMAS, developed at the University of Dortmund by Müller-Clostermann [60], is a textual tool for Markovian queueing network analysis. As an extension, NUMAS allows the modeling of queues with server breakdowns and repairs. NUMAS thus allows for steady state performability analysis.

The graphical performance analysis tool MACOM, developed by Sczittnick *et al.* at the University of Dortmund [73], is mainly used for the steady state analysis of blocking phenomena in communication networks. MACOM emphasizes advanced techniques for the steady state analysis of large MRMs.

Based on PRSs [32] are the tools METFAC, ASSIST and USENUM. The textual tool METFAC, developed by Carrasco and Figueras at the University of Catalunya [10, 11], supports the use of a PRS specification technique and has been used for performance, dependability and performability modeling of computer systems. Steady state, transient, as well as cumulative measures can be computed.

The tool ASSIST has been developed by Johnson and Butler at NASA [39] as a front-end to the SURE package [8] for reliability analysis of (computer) systems. This textual tool allows for the flexible specification of PRS. By the use of arrays of state variables and loops in the production rules, compact specifications can be written. Also facilities for truncating state spaces are available. The ASSIST program translates the PRS to input for the SURE package. This input is an MRM. The SURE package deals with absorbing semi-Markov models. Therefore, only transient measures are computed.

The textual tool USENUM, developed by Sczittnick *et al.* at the University of Dortmund [7, 72], allows users to define Markovian models by means of a finite state machine. USENUM can be used stand-alone, or within the QN tool MACOM.

4.3 Stochastic Petri net models

A wide variety of tools for stochastic Petri nets have been developed. We briefly discuss the most well-known tools that are based on MRMs.

GreatSPN, developed by Chiola *et al.* at the University of Torino [12], is a graphical Petri net tool which is primarily used for the performance analysis of computer and communication systems. Analysis techniques are mainly for steady state measures.

ESP, developed by Bobbio and Cumani [6, 21], is a textual SPN tool. In this tool, special emphasis is put on the use of phase-type distributions instead of only exponential distributions, on transient measures and on the aggregation of stiff MRMs.

METASAN, developed by Sanders and Meyer [68, 69] at the University of Michigan, is based on SANs. The tool includes steady state, transient and cumulative analysis methods.

The tool UltraSAN, developed by Sanders *et al.* [20] at the University of Arizona, is also based on the SAN concept. With UltraSAN, the input of the models is totally graphical. UltraSAN allows for a structured form of hierarchical modeling which results in lumped underlying MRMs that are substantially smaller (so-called reduced base-models [70]) than their "flat" counterparts. Steady state as well as transient simulation are also available as solution methods.

SPNP, developed by Ciardo *et al.* [14], is a C-based SPN tool which allows for a flexible definition of a class of SPN models known as stochastic reward nets. Steady state, transient and cumulative measures are supported. By the flexible use of C, it is possible to construct models hierarchically, that is, results of one model can be used in the analysis of another model, even in a fixed point iterative manner [17].

TOMSPIN is a general SPN tool developed at SIEMENS AG [41], for performance and dependability analysis. Steady state and transient measures are supported. An approximate solution for hierarchically structured SPN models based on an aggregation algorithm is also included.

PENPET is a performability modeling tool developed by Lepold [42, 43] at SIEMENS AG. It is a high-level tool built on top of TOMSPIN in which one SPN is used for the specification of system dependability aspects, and another for the system performance aspects.

The graphical tool DSPNexpress has been developed by Lindemann at the Technical University of Berlin [46]. Interesting aspect of this tool is that it allows for DSPNs, i.e., SPNs in which transitions may have deterministic timing. Under certain conditions, an embedded Markov chain can be constructed that allows one to solve for the steady state probabilities.

4.4 Hierarchical and Approximate Models

The performability modeling tools DyQNtool$^+$ [35] has been developed by Haverkort *et al.* at the University of Twente. The tool operates along the lines of the dynamic queueing network concept and is an extension of its predecessor DyQNtool

[28, 29, 30]. DyQNtool+ has been developed as a shell of programs around the packages SHARPE and SPNP. The SHARPE package is used for the solution of series of PFQNs of which the results are used as reward rates in the SRNs specified using SPNP. Thus, in a very flexible way, performability models can be evaluated.

The tool SHARPE has been developed by Sahner and Trivedi [66, 67]. SHARPE allows users to specify SPN, QN and FT like models as well as MRMs directly. Also hierarchical modeling is possible, that is, the results of a model analysis can be used in higher-level model evaluations, possibly using a different modeling approach.

5 Problems and perspectives

In this section, we discuss a number of recurring problems in performance, dependability and performability evaluation. We discuss the issue of largeness in Section 5.1 and the issue of stiffness in Section 5.2. The modeling of non-exponential behavior is addressed in Section 5.3.

5.1 Largeness

Models of practical systems are often very large. We use special specification, generation, storage and solution techniques to deal with the large models (largeness tolerance) or avoid largeness altogether.

First consider the techniques related to largeness tolerance. With this we mean the techniques that aim at being able to handle models as large as possible without affecting the model size itself.

For non-state space models, largeness tolerance techniques would encompass better implementations of MVA and convolution like algorithms, using extra significant digits to keep the normalizing constants accurate. Sparse storage techniques should be used whenever matrices are involved, such as in MGM. Also, whenever possible, special properties should be exploited. As an example of this, the matrix $\mathbf{R}$ that has to be calculated when using MGM has the property that zero entries come in rows, i.e., whenever the first element of a row equals zero, the rest is zero as well. This can be exploited in devising the storage and computational schemes, as has been done in Xmgmtool [34].

For MRM-based modeling techniques, largeness tolerance techniques presuppose the use of a concise specification method (e.g. SPN), automated MRM generation, sparse storage, sparsity preserving numerical techniques, e.g., using SOR instead of Gaussian elimination for the solution of the steady-state behavior of large MRMs, and orthogonal uniformization for the transient solution of acyclic Markov chains [27].

With largeness avoidance, we try to circumvent the generation of very large models. Although the capacity of modern day workstations is enormous, there will always remain systems that yield models that become too large to be handled directly.

As mentioned before, an important largeness avoidance technique widely applicable is hierarchical modeling which is based on the "divide and conquer" principle. The basic idea is to split a large model in smaller ones that can be analyzed in isolation. The results of the submodels are integrated in a single overall model that is small enough to be analyzed.

In the field of PFQN, it has been shown that such a decomposition approach can be performed in an exact way ("Norton's theorem", due to Chandy, Herzog and Woo [71]). For non-PFQNs, the decomposition is approximate. The theory developed by Courtois and which is used for the analysis of the degradable QN in NUMAS establishes bounds on the error made in the steady-state probabilities thus obtained. In fact, the performability evaluation approach based on MRM is motivated by these decomposition properties. An approximate way of solving large SPN models by decomposition is discussed by Ciardo and Trivedi [17]. They use a fixed-point iteration scheme. In this context, the work on automatic lumping as performed in UltraSAN by Sanders *et al.* is also of interest [20, 70].

The truncation of "the least important states", i.e., those states that have a small probability mass is another way to avoid large models. Work in this direction has been done by Haverkort [33], Li and Silvester [44], Muppala *et al.* [61], Bavuso *et al.* [4] and de Souza e Silva and Ochoa [75]. It is especially appropriate in case MRM or SPN models are used.

5.2 Stiffness

Informally speaking, stiffness is a property of a model to take very long to be solved. Often this is caused by the fact that in the model parameters of widely varying order of magnitude play an important role; this clearly is the case in dependability-related models where failure rates are very small and repair rates are orders of magnitudes larger.

Given a particular model type, one can specialize the above informal definition of stiffness. For MRMs, stiffness is often defined as ratio between the largest and smallest rate in the transition rate matrix; the higher this ratio, the more stiff the model. Even more refined are the definitions related to a solution technique for a specific model. In using uniformization for instant-of-time measures of MRMs, the value of qt is often called the stiffness index, where t is the time epoch of interest and q is the maximum over the absolute values of the diagonal entries of the rate matrix, i.e., the uniformization rate. The recently developed extensions of the uniformization technique such as steady-state detection [62] and adaptive uniformization [58] decrease the impact of stiffness.

For very stiff MRMs, implicit integration techniques (such as Runge-Kutta) seem to be the most efficient [48]. The use of these techniques in combination with uniformization also seems fruitful [48].

When discrete-event simulation is the used as a solution method, stiffness can often be circumvented by using importance sampling [26] as implemented in SAVE [25] and UltraSAN [20] or by using injection simulation [57].

Model decomposition, where the fast and slow rates are separated from each other, is another way of avoiding stiff models. This is implicitly done in many hierarchical solution techniques [23].

5.3 Non-exponential behavior

In many model solution techniques, the only allowed time-distributions are exponential distributions. This is the case for the FCFS stations in PFQNs and for the timed transitions in SPNs. To include more variability in a model is generally not much of a problem since a hyper-exponential distribution with two phases can be used to create very large coefficients of variation. More of a problem is the inclusion of (quasi-)deterministic timing.

When simulation is used as a solution technique, non-exponential timing is not a problem. Moreover, it often reduces simulation time as less variance is put in.

One common way to "Markovize" a non-exponential distribution is to use phase-type distributions. In combination with MGMs this is a very attractive approach, although it increases the size of the models to be dealt with. In the context of SPN models or MRMs, this method of phases is often less easy to apply: the state space suffers tremendously.

Recent developments in the field of DSPNs (introduced by Ajmone Marsan and Chiola [1] and further developed by Lindemann [45]) and Markov-regenerative SPNs, introduced by Choi *et al.* [13], alleviate the exponential assumption in SPN models significantly. When one deals with product-form SPNs [36], insensitivity properties known from stochastic-processes theory establish that in many circumstances it does not really matter what the form of the distributions is; only their means matter.

6 Future Work and Concluding Remarks

The challenge in the future lies in developing modeling tools that will operate in a highly constrainted and schedule-oriented environment. The availability of such tools will provide a means of effectively integrating performance experts into the product design team.

In the future, tools must be made sophisticated enough so that an expert system might even eliminate the need for the human expert. Then the designer himself could perform the analyses without any special expertise in the theories of reliability, queues, Markov chains and stochastic Petri nets. At the very least, it would help to smooth variations in the competence of the analyst from one group to the next. A performance/reliability 'center of competency' department, that provides upgrades and maintenance to and advice on, the tools itself might then arise; but the actual analysis would shift to the designers themselves.

A great deal of progress has been made in the last decade in techniques for the generation and solution of large performance, reliability and performability models. Correspondingly, software tools have also been built and distributed. Due to the increased capacity of modern-day workstations, mathematical evaluation techniques for performability and dependability evaluation have become

much more feasible. Because modern-day workstations have large internal memories (up to a few hundred MB) and very fast processors (at least 50 MHz clock frequency) the numerical evaluation of large to very large models has become possible. Also, simulation experiments that were unthinkable are now within reach by using only moderately priced workstations.

However, the need to construct and evaluate even larger models continues. Modern computer-communication systems have reached such a complexity that evaluation of their performability and dependability during the design process is an absolute necessity in order to build high-performance systems that provide the requested service for a reasonable price.

Despite the above need, there is still a long way to go to a really integrated design-evaluation path. Still, a lot of progress has been made over the last two decades. In this paper we overviewed that progress and indicated some issues we think will be of importance in the coming years.

References

1. M. Ajmone-Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Lecture Notes in Computer Science*, volume 266, pages 132–145. Springer-Verlag, 1987.
2. M. Ajmone Marsan, G. Conte, G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems", *ACM Transactions on Computer Systems* **2**(2), pp.93–122, 1984.
3. G. Balbo, S.C. Bruell, S. Ghanta, "Combining Queueing Networks and Stochastic Petri Nets for the Solution of Complex Models of System Behaviour", *IEEE Transactions on Computers* **37**(10), pp.1251–1268, 1988.
4. S.J. Bavuso, J. Bechta Dugan, K.S. Trivedi, E.M. Rothmann, W.E. Smith, "Analysis of Typical Fault-Tolerant Architectures using HARP", *IEEE Transactions on Reliability* **36**(2), pp.176–185, 1987.
5. H. Beilner, J. Mäter, N. Weissenberg, "Towards a Performance Modelling Environment: News on HIT", in: *Modelling Techniques and Tools for Computer Performance Evaluation*, Editors: D. Potier, R. Puigjaner, Plenum Press, pp.57–75, 1989.
6. A. Bobbio, "Petri Nets Generating Markov Reward Models for Performance/Reliability Analysis of Degradable Systems", in: *Modelling Techniques and Tools for Computer Performance Evaluation*, Editors: D. Potier, R. Puigjaner, Plenum Press, pp.353–365, 1989.
7. P. Buchholz, *Die strukturierte Analyse Markoffscher Modelle, Informatik Fachberichte* 282, Springer Verlag, 1991.
8. R.W. Butler, "The SURE Reliability Analysis Program", *NASA Technical Memorandum* 87593, 1986.
9. J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Commun. ACM.*, 16(9):527–531, Sept. 1973.
10. J.A. Carrasco, *Modelacion y Evaluacion de la Tolerancia a Fallos de Sistemas Distribuidos con Capacidad de Reconfiguracion*, PhD thesis, University of Catalunya, Spain, 1986.
11. J.A. Carrasco, J. Figueras, "Metfac: Design and Implementation of a Software Tool for Modeling and Evaluation of Complex Fault-Tolerant Computing Systems", *Proceedings FTCS 16*, IEEE Computer Society Press, pp.424–429, 1986.

12. G. Chiola, "A Graphical Petri Net Tool for Performance Analysis", in: *Modelling Techniques and Performance Evaluation*, Editors: S. Fdida, G. Pujolle, North-Holland, pp.323–333, 1987.
13. H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov Regenerative Stochastic Petri Nets. In *16th IFIP W.G. 7.3 Int'l Sym. on Computer Performance Modelling, Measurement and Evaluation (Performance'93)*, Rome, Italy, Sep. 1993.
14. G. Ciardo, J. Muppala, K.S. Trivedi, "SPNP: Stochastic Perti Net Package", *Proceedings of the Third International Workshop on Petri Nets and Performance Models*, IEEE Computer Society Press, pp.142–151, 1989.
15. G. Ciardo, J. Muppala, and K. Trivedi, "Analyzing Concurrent and Fault-Tolerant Software using Stochastic Reward Nets," *Journal of Parallel and Distributed Computing*, Vol. 15, pp. 255-269, 1992.
16. G. Ciardo, A. Blakemore, P.F.J. Chimento, J.K. Muppala, K.S. Trivedi, "Automated Generation and Analysis of Markov Reward Models using Stochastic Reward Nets", in: *Linear Algebra, Markov Chains, and Queueing Models*, Editors: C. Meyer and R. J. Plemmons, Vol.48 of *IMA Volumes in Mathematics and its Applications*, Springer-Verlag, 1992.
17. G. Ciardo, and K. S. Trivedi, " Decomposition Approach for Stochastic Reward Net Models," *Performance Evaluation*, Vol. 18, No. 1, pp. 37-59, July 1993.
18. J. L. Coleman, W. Henderson, P. G. Taylor, *Product Form Equilibrium Distributions and a Convolution Algorithm for Stochastic Petri Nets* Research Report, University of Adelaide, 1992
19. A. J. Coyle, W. Henderson, P. G. Taylor, "Reduced Load Approximations for Loss Networks", to appear in Telecommunications Systems.
20. J. A. Couvillion, R. Freire, R. Johnson, W.D. Obal II, A. Qureshi, M. Rai, W.H. Sanders, J.E. Tvedt, "Performability Modelling with UltraSAN", *IEEE Software*, pp.69–80, September 1991.
21. A. Cumani, "ESP—A Package for the Evaluation of Stochastic Petri Nets with Phase-Type Distributed Transition Times", *Proceedings of the International Workshop on Timed Petri Nets*, IEEE Computer Society Press, pp.144–151, 1985.
22. J. Bechta Dugan, R. Geist and M. Smotherman, "The Hybrid Automated Reliability Predictor", *AIAA Journal on Guidance, Control and Dynamics*, Vol. 9, No. 3, May-June 1986, pp. 319-331.
23. R. Geist, M. Smotherman, K. S. Trivedi, J. Bechta Dugan, "Reliability Analysis of Life-Critical Systems", *Acta Informatica*, Vol. 23, No. 6, Nov. 1986.
24. R. Geist, K.S. Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques", *IEEE Computer* 23(7), pp.52–61, 1990.
25. A. Goyal, W.C. Carter, E. de Souza e Silva, S.S. Lavenberg, K.S. Trivedi, "The System Availability Estimator", *Proceedings FTCS 16*, IEEE Computer Society Press, pp.84–89, 1986.
26. A. Goyal, P. Heidelberger, and P. Shahabuddin. Measure specific dynamic importance sampling for availability simulations. In A. Thesen, H. Grant, and W. D. Kelton, editors, *Proc. of the 1987 Winter Simulation Conference*, 1987.
27. D. Gross, D.R. Miller, "The Randomization Technique as a Modelling Tool and Solution Procedure for Transient Markov Processes", *Operations Research* **32**(2), pp.343–361, 1984.
28. B.R. Haverkort, I.G. Niemegeers, "Using Dynamic Queueing Networks as a Tool for Specifying Performability Models", *ACM Performance Evaluation Review* **17**(1), p.225, 1989.

29. B.R. Haverkort, *Performability Modelling Tools, Evaluation Techniques, and Applications*, Ph.D. thesis, University of Twente, 1990.
30. B.R. Haverkort, I.G. Niemegeers, P. Veldhuyzen van Zanten, "DyQNtool—A Performability Modelling Tool Based on the Dynamic Queueing Network Concept", in: *Computer Performance Evaluation: Modelling Techniques and Tools*, Editors: G. Balbo, G. Serazzi, North-Holland, pp.181–195, 1992.
31. B.R. Haverkort, "Approximate Performability Modelling using Generalized Stochastic Petri Nets", *Proceedings of the 1991 International Workshop on Petri Nets and Performance Models*, IEEE Computer Society Press, 1991, pp.300-309.
32. B. Haverkort and K. Trivedi. Specification and generation of Markov reward models. *Discrete-Event Dynamic Systems: Theory and Applications* 3 , pp.219–247, 1993.
33. B.R. Haverkort, "Approximate Performability and Dependability Modelling using Generalized Stochastic Petri Nets", *Performance Evaluation* **18**(1), pp.61–78, 1993.
34. B. R. Haverkort, A.P.A. van Moorsel, and D-J Speelman. Xmgm: Performance modeling using matrix geometric techniques. In *Proceedings of the 2nd Int'l workshop on modeling, analysis and simulation of computer and telecommunication systems*, 1994.
35. B.R. Haverkort, "Performability Evaluation using DyQNtool$^+$", submitted for publication, 1994.
36. W. Henderson and P. G. Taylor, "Aggregation Methods in exact performance analysis of stochastic Petri nets", *Proceedings of the 3rd Int'l Workshop on Petri Nets and Performance Models*, pp.12–18, 1989.
37. R.A. Howard, *Dynamic Probabilistic Systems, Vol. II: Semi-Markov and Decision Processes*, New York, Wiley, 1971.
38. A.M. Johnson Jr., M. Malek, "Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability", *ACM Computing Surveys* **20**(4), pp.227–269, 1988.
39. S.C. Johnson, R.W. Butler, "Automated Generation of Reliability Models", *Proceedings of the 1988 Annual Reliability and Maintainability Symposium*, pp.17–22, 1988.
40. V. 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 Intl. Symp. on Fault Tolerant Computing*, Vienna, Austria, July 1986. IEEE.
41. R. Lepold, "Tomspin: Benutzerhandbuch", internal report Siemens AG, 1991.
42. R. Lepold, "PENPETt: A New Approach to Performability Modelling using Stochastic Petri Nets", *Proceedings of the First International Workshop on Performability Modelling of Computer and Communication Systems*, Editors: B.R. Haverkort, I.G. Niemegeers, N.M. van Dijk, University of Twente, pp.3–17, 1991.
43. R. Lepold, "Performability Evaluation of a Fault-Tolerant Computer Systems using Stochastic Petri Nets", *Proceedings of the Fifth International Conference on Fault-Tolerant Computing Systems*, Springer Verlag, Nürnberg, 1991.
44. V.O.K. Li, J.A. Silvester, "Performance Analysis of Networks with Unreliable Components", *IEEE Transactions on Communications* **32**(10), pp.1105–1110, 1984.
45. C. Lindemann, "An improved numerical algorithm for calculating steady-state solutions of deterministic and stochastic Petri net models", *Proceedings of the 4th Int'l Workshop on Petri Nets and Performance Models*, 1991.
46. C. Lindemann, R. German, "DSPNexpress: A Software Package for Efficiently Solving Deterministic and Stochastic Petri Nets", in: *Performance Tools 1992*, Editors: R. Pooley, J. Hillston, Edinburgh University Press Ltd., forthcoming, 1992.

47. E. A. MacNair and C. H. Sauer. *Elements of practical performance modeling.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1985.
48. Manish Malhotra *Specification and Solution of Dependability Models of Fault-Tolerant Systems*, Ph.D. Thesis, Dept. of Comp. Sc., Duke University, April 1993.
49. R. A. Marie. An approximate analytical method for general queueing networks. *IEEE Trans. Software Engg.*, SE-5:530–538, 1979.
50. R. A. Marie, A. L. Reibman, K. S. Trivedi "Transient Analysis of Acyclic Markov Chains", *Performance Evalution* **7**, 1987.
51. J.F. Meyer, "On Evaluating the Performability of Degradable Computer Systems", *IEEE Transactions on Computers* **29**(8), pp.720–731, 1980.
52. J.F. Meyer, "Closed-Form Solutions of Performability", *IEEE Transactions on Computers* **31**(7), pp.648–657, 1982.
53. J.F. Meyer, A. Movaghar, W.H. Sanders, "Stochastic Activity Networks: Structure, Behavior, and Application", *Proceedings of the International Workshop on Timed Petri Nets*, IEEE Computer Society Press, pp.106–115, 1985.
54. J.F. Meyer, "Performability: A Retrospective and Some Pointers to the Future", *Performance Evaluation*, **14**(3&4), pp.139–156, 1992.
55. K. B . Misra (Ed.), *New Trends in System Reliability Evaluation*, Elsevier Science Publishers, 1993.
56. M.K. Molloy, "Performance Analysis using Stochastic Petri Nets", *IEEE transactions on Computers* **31**(9), pp.913–917, 1982.
57. A.P.A. van Moorsel, *Performability Evaluation Concepts and Techniques*, Ph.D. thesis, University of Twente, Department of Computer Science, 1993.
58. A.P.A. van Moorsel, W.H. Sanders, "Adaptive Uniformization", forthcoming in *Stochastic Models*, 1994.
59. M. Mulazzani, K.S. Trivedi, "Dependability Prediction: Comparison of Tools and Techniques", *Proceedings IFAC SAFECOMP*, pp.171–178, 1986.
60. B. Müller-Clostermann, "NUMAS—A Tool for the Numerical Analysis of Computer Systems", in: *Modelling Techniques and Tools for Computer Performance Analysis*, Editor: D. Potier, North-Holland, pp.141–154, 1985.
61. J. K. Muppala, A. Sathaye, R. Howe, K. S. Trivedi, "Dependability Modeling of a Heterogeneous VAXcluster System Using Stochastic Reward Nets", in: *Hardware and Software Fault Tolerance in Parallel Computing Systems*, Editor: D. Averesky, Ellis Horwood Ltd., 1992, forthcoming.
62. J. Muppala and K. S. Trivedi, "Numerical Transient Solution of Finite Markovian Queueing Systems", in: *Queueing and Related Models,* U. N. Bhat and I. V. Basawa (ed.), pp. 262-284, Oxford University Press, 1992.
63. D. Potier, M. Veran, "The Markovian Solver of QNAP2 and Examples", in: *Computer Networking and Performance Evaluation*, Editors: T. Hasegawa, H. Takagi, Y. Takahashi, pp.259–279, 1986.
64. A.L. Reibman, K.S. Trivedi, "Transient Analysis of Cumulative Measures of Markov Model Behavior", *Stochastic Models* **5**(4), pp.683–710, 1989.
65. M. Reiser and S. S. Lavenberg. Mean value analysis of closed multichain queueing networks. *J. ACM.*, 27(2):313–322, Apr. 1980.
66. R.A. Sahner, K.S. Trivedi, "Reliability Modelling using SHARPE", *IEEE Transactions on Reliability* **36**(2), pp.186– 193, 1987.
67. R.A. Sahner, K.S. Trivedi, "A Software Tool for Learning About Stochastic Models", *IEEE Transactions on Education* **36**(1), 1993.
68. W.H. Sanders, J.F. Meyer, "Performability Evaluation of Distributed Systems using Stochastic Activity Networks", *Proceedings of the 1987 International Workshop*

on Petri Nets and Performance Models, IEEE Computer Society Press, pp.111–120, 1987.

69. W.H. Sanders, *Construction and Solution of Performability Models Based on Stochastic Activity Networks*, Ph.D. dissertation, University of Michigan, USA, 1988.
70. W.H. Sanders, J.F. Meyer, "Reduced Base Model Construction for Stochastic Activity Networks", *IEEE Journal on Selected Areas in Communications* **9**(1), pp.25–36, 1991.
71. C. H. Sauer and K. M. Chandy. *Computer Systems Performance Modeling.* Prentice-Hall, 1981.
72. M. Szczittnick, *Techniken zur funktionalen und quantitativen Analyse von Markoffschen Rechensystemmodellen*, M.Sc. thesis, University of Dortmund, August 1987.
73. M. Szczittnick, B. Müller-Clostermann, "MACOM—A Tool for the Markovian Analysis of Communication Systems", in: *Proceedings of the Fourth International Conference on Data Communication Systems and Their Performance*, Editor: R. Puigjaner, pp.456–470, 1990.
74. E. de Souza e Silva, H.R. Gail, "Performability Analysis of Computer Systems: from Model Specification to Solution", *Performance Evaluation*, **14**(3&4), pp.157–196, 1992.
75. E. de Souza e Silva, P.M. Ochoa, "State Space Exploration in Markov Models", *ACM Performance Evaluation Review* **20**(1), pp.152–166, 1992.
76. M.F. Squillante, "MAGIC: A Computer Performance Modelling Tool based on Matrix-Geometric Techniques", in: *Computer Performance Evaluation: Modelling Techniques and Tools*, Eds.: G. Balbo, G. Serazzi, North-Holland, pp.411–425, 1992.
77. W. J. Stewart, "MARCA: Markov Chain Analyzer", in: *Numerical Solution of Markov Chains*, Editor: W.J. Stewart, Marcel Dekker, 1991.
78. L. Tomek, V. Mainkar, R. Geist, and K. Trivedi. Reliability analysis of life-critical real-time systems. *Proceedings of the IEEE*, January 1994.
79. K.S. Trivedi, J.K. Muppala, S.P. Woolet, B.R. Haverkort, "Composite Performance and Dependability Analysis", *Performance Evaluation*, **14**(3&4), pp.197–215, 1992.
80. M. Veran, D. Potier, "QNAP2: A Portable Environment for Queueing System Modelling", in: *Modelling Techniques and Tools for Computer Performance Evaluation*, Editor: D. Potier, North-Holland, pp.25–63, 1985.
81. W. Whitt. The queueing network analyzer. *The Bell System Technical Journal*, 62(9), Nov. 1983.