ELSEVIER

Performance Evaluation 25 (1996) 17–40

PERFORMANCE EVALUATION
An International Journal

Performability modelling tools and techniques

Boudewijn R. Haverkort *, Ignas G. Niemegeers

Tele-Informatics and Open Systems, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Received 1 June 1992; revised 8 April 1994

Abstract

Over the last decade considerable effort has been put in the development of techniques to assess the performance and the dependability of computer and communication systems in an integrated way. This so-called performability modelling becomes especially useful when the system under study can operate partially, which is for instance the case for fault-tolerant computer systems and distributed systems.

Modelling techniques are a fundamental prerequisite for actually doing performability analysis. A prerequisite of a more practical but not less important nature is the availability of software tools to support the modelling techniques and to allow system designers to incorporate the new techniques in the design process of systems.

Since performability modelling requires many aspects of a system to be specified, high requirements should be posed on performability modelling tools. Moreover, these tools should be structured such that the models can be specified at a level that is easy to understand for a system designer, and that the mathematical aspects are hidden as much as possible. The output of the tool should also be such that it can be understood with only limited knowledge of the underlying mathematical model.

We have developed a new, fairly general modelling tool framework that can be used as a guide to assess the usability and structure of performability modelling tools. After briefly reviewing the mathematical aspects of performability modelling we discuss this framework. We then discuss 12 recently developed tools (Metaphor, Numas, Metasan, Metfac, Save, Sharpe, SPNP, Tangram, Penpet, UltraSAN, Surf-2, DyQNtool$^+$) that can all be used for some aspects of performability modelling and analysis. We assess among other things their structure, their capabilities in terms of measures that can be obtained, and the used modelling formalism. We also discuss directions for future work in the field of performability modelling tools.

Keywords: Dependability; General modelling tool framework; Markov reward models; Performability; Performance; Queueing networks; Stochastic Petri nets

1. Introduction

Over the last decade there has been an increased interest in the integrated modelling of performance and dependability aspects [1] of computer and communication systems. This so-called performability modelling

* Corresponding author. Current address: Department of Computer Science — Distributed Systems, Rheinisch-Westfälische Technische Hochschule Aachen, D-52056 Aachen, Germany. E-mail: haverkort@informatik.rwth-aachen.de.

[1] Note that we use the term *dependability* here to denote either reliability or availability. In the original definition by Laprie security and safety aspects are also included [47]. We do not address these aspects here.

SSDI 0166-5316(94)00038-7

was motivated by the fact that in many modern computer and communication systems, the servicing of jobs can continue, even in the presence of failures. Examples of such systems are fault-tolerant computer systems, parallel computer systems, and distributed computer systems. For all these systems the failure of a specific component does not necessarily affect the overall service provided by the system, but it does affect the speed at which the overall service can be provided.

Since more and more applications depend on the correct and timely operation of systems of the above classes (airline reservation systems, banking systems, aircraft control systems, integrated office systems, to name just a few), the analysis of the performance of these systems in the presence of failure (and repairs) is of major importance. Moreover, important users of these types of systems such as telephone and banking companies, often require that suppliers give evidence that their systems can provide a particular quality of service over a certain time span, even in the presence of failures.

With the integration of computer systems and telecommunications (telematics or tele-informatics) the quality of service (QoS) concept as often used in recommendations by standardization bodies such as CCITT will become more important as a measure to qualify systems. Moreover, this QoS measure as defined by CCITT must reflect the combined influence of dependability and performance associated factors (Recommendation G.106) [11]. It seems that the concept of performability can be used to handle such QoS look-alike measures, thus making it even more important for the future [58].

For the above reasons performability modelling and analysis have received much attention. The emphasis, however, has mostly been on the mathematical aspects of performability modelling, i.e. the analysis of Markov reward models. Although important and a prerequisite for doing performability analysis, the mathematical aspects are not the only important ones. The issue of constructing performability models in a convenient way is also of key importance. Moreover, we think that performability modelling will not be accepted as a "normal" analysis technique as performance modelling is nowadays, as long as no good software tools are available that help in the construction of the models. A number of tools that are in some way suited for performability modelling and analysis have been reported in the literature recently. The aim of this paper is to survey and compare these tools. Aspects that we address are, amongst others, the class of models that can be handled by the tools, their user-interface and their modelling language.

Some of the tools discussed in this paper have been addressed in other survey papers as well. In the papers by Johnson and Malek [44] and Mulazzani and Trivedi [61] the emphasis lies solely on tools for dependability modelling and analysis. Meyer [33,58] describes the historic development of performability, thereby briefly addressing a few tools, in far less detail than we do here. In their paper [88], de Souza e Silva and Gail emphasize randomization based performability evaluation techniques. They also address a few tools but, apart from the tool Tangram, again in far less detail as we do here. Haverkort and Trivedi [38] discuss tools and techniques for the construction of Markov reward models in general, not specifically emphasizing on performability modelling. In this paper, for the first time, a systematic overview of a large number of performability modelling tools is given.

This paper is organized as follows. In Section 2 we briefly discuss the mathematical background of performability modelling as far as this is necessary to indicate differences and similarities in the various tool capabilities. Then, in Section 3 we discuss a fairly general modelling tool framework that turns out to be useful for the classification of the various tools. In Section 4, the main part of this paper, we discuss twelve tools for performability modelling and analysis. In Section 5 we summarize the paper and give some directions for future work in the area of performability modelling tools.

2. Mathematical aspects of performability modelling

Gracefully degradable computer systems are assumed to be able to perform at various levels of performance. At system start all components are assumed to be operational and the system will operate at maximal performance. When a component fails, the system will reconfigure itself and restart its activities, albeit with degraded performance. Due to the fact that time intervals between failures are in general relatively large, it is usually assumed that the system will be in steady state most of the time between successive reconfigurations and failures.

Let $\mathcal{X}$ denote the set of all possible configurations in which the system can operate. Now, define a continuous-time stochastic process $X = (X_t, t \geq 0)$, $X_t \in \mathcal{X}$, describing the structure of the system at time t, i.e. which components are up and which are down at time t. X is often referred to as the *structure-state process* since it describes the state of the system structure. The (steady-state) performance of the system when in structure state $i \in \mathcal{X}$ is denoted by $r(i)$, where $r : \mathcal{X} \to \mathbb{R}$ is a real valued *reward rate function* on the state space $\mathcal{X}$. Note that the function r has to be defined by multiple performance analyses, i.e. for every $i \in \mathcal{X}$ corresponding to some system configuration, the value $r(i)$, e.g. the throughput, has to be obtained with a "classical" performance analysis. The values $r(i)$ summarize the performance of the system in structure state i, instead of coping with all possible performance states in every possible structure state. This decomposition, i.e. the separate analysis of the performance given a particular structure state and the subsequent Markov reward analysis, is based on near-complete decomposability arguments as discussed by Courtois [19].

For $i \in \mathcal{X}$, let the row vector $\boldsymbol{\pi} = [\ldots, \pi_i, \ldots]$ denote the initial probability vector on $\mathcal{X}$, the row vector $\boldsymbol{p} = [\ldots, p_i, \ldots]$ the steady-state probability of residing in state i, and the row vector $\boldsymbol{p}(t) = [\ldots, p_i(t), \ldots]$ the (transient) probability of residing in i at time t. The following measures can be distinguished:

(1) steady-state performability (SSP): $P = \sum_{i \in \mathcal{X}} p_i r(i)$;
(2) the transient or point performability (TP): $P(t) = \sum_{i \in \mathcal{X}} p_i(t) r(i)$;
(3) the mean reward to absorption (MRTA): $\text{MRTA} = \sum_{i \in \mathcal{X}_A} z_i r(i)$ ($\mathcal{X}_A \subset \mathcal{X}$ and z_i are defined below);
(4) the cumulative performability (CP): $Y(t) = \int_0^t r(X_s)\,\mathrm{d}s$;
(5) the performability distribution (PDF): $F(t, y) = \Pr\{Y(t) \leq y\}$.

The MRTA is only defined for models with an absorbing state (see below). It should be noted that there is no consensus about whether all the above measures are performability measures. Some authors require performability measures to be at least time dependent, thus excluding the steady-state performability measures (category (1) above). Originally, only the CDF of the cumulative reward (category (4) above) was addressed and called *the performability* [55,56,58].

If one chooses $r(i) = 1$ if the system is operational in state $i \in \mathcal{X}$ and $r(i) = 0$ otherwise, then $E[Y(t)]$ is simply the cumulative uptime. The fraction $E[Y(t)]/t$ is then the availability (sometimes referred to as the interval availability as opposed to the point availability, the latter being a particular case of the point performability measures: simply $E[X(t)]$ given the above reward assignment).

The basic model discussed so far is a so-called *rate-based* reward model which means that when residing in a particular structure state $i \in \mathcal{X}$ at time t, the system performs with rate $r(i)$. The rates, however, may also depend on the global time t, thus having a reward rate function $r(i; t)$ for every state $i \in \mathcal{X}$. It is also possible to address *impulse-based* reward models. With these models a *reward impulse function* $r : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ has to be defined which associates a reward $r(i, j)$ with every transition from state $i \in \mathcal{X}$

to state $j \in \mathcal{X}$. Every time a transition from state i to state j takes place, the cumulatively obtained reward increases instantaneously with $r(i, j)$ units. Of course these rewards may also depend on the global time thus having transition reward functions $r(i, j; t)$. In general, combinations of these four possibilities may coexist. For an overview of various Markov reward models the reader is referred to Howard [42].

In the rest of this paper we restrict our discussion to Markov reward models. With a few exceptions only rate-based models are addressed. When the structure-state process is a continuous-time Markov chain with generator matrix $\boldsymbol{Q}$ then the computation of the steady-state measures requires the solution of the following system of linear equations:

$$\boldsymbol{pQ} = \boldsymbol{0} \quad \text{and} \quad \boldsymbol{pe}^{\mathrm{T}} = 1, \tag{1}$$

where $\boldsymbol{0}$ is a row vector of 0's and $\boldsymbol{e}^{\mathrm{T}}$ a column vector of 1's. This system of linear equations can be solved directly, e.g. by Gaussian elimination, or iteratively by Gauss–Seidel iteration or by successive over-relaxation (SOR). The former is the most convenient for small systems, the latter for larger systems. For comparisons, we refer to [46,90].

When the CTMC contains an absorbing state, a steady-state measure does not provide useful information. Instead, the expected time or reward until absorption then is of interest. Let $\boldsymbol{Q}_A$ equal the matrix $\boldsymbol{Q}$, however, restricted to the non-absorbing states $\mathcal{X}_A \subset \mathcal{X}$. A similar definition can be given for π_A. The row vector z, which is the solution of

$$z\boldsymbol{Q}_A = -\pi_A, \tag{2}$$

now represents the amount of time spent in every non-absorbing state before absorption. The mean reward to absorption then equals: MRTA $= \sum_{i \in \mathcal{X}_A} z_i r(i)$. The system of linear equations (2) can be solved with similar means as system (1).

The transient (or point) performability measures require the solution of the following system of linear differential equations:

$$\boldsymbol{p}'(t) = \boldsymbol{p}(t)\boldsymbol{Q} \quad \text{with } \boldsymbol{p}(0) = \pi. \tag{3}$$

Semi-symbolic solutions for this system of differential equations are computationally expensive [53,69] and therefore are only applicable for small models. For larger models, numerical techniques are used. One can either use Runge–Kutta based methods, possibly adapted for stiff systems [70–72], or randomization (also called uniformization) [31,32,59,64,86–88]. For the computation of $E[Y(t)]$ similar techniques can be employed.

For the CDFs of the cumulative measures various specialized algorithms exist [22,26,67,68]. For acyclic Markov chains recursive algorithms exist. For general Markov chains algorithms are mostly based on randomization [87,88] or on Laplace [85] or Laguerre transforms [1]. In the latter cases, the back-transformation to the time domain is mostly done in an approximate fashion.

Recently, a number of survey papers on performability analysis techniques have been published in a special issue of *Performance Evaluation* [21]. We refer to [58] in which the history of the development of performability modelling is sketched; to [88] in which performability modelling and evaluation are discussed with an emphasis on randomization and object-oriented model specification; and, to [92] in which various forms of combined performance and dependability modelling are discussed.

3. A general modelling tool framework

As is clear from Section 2 there are many evaluation techniques for obtaining specific measures from Markov reward models. Most of these techniques make extensive use of numerical analysis and can therefore only be used when computer support is available. However, even when computer support is available for the numerical analysis there are a number of problems that hinder the use of the state-of-the-art Markov reward model evaluation techniques:

(1) system designers are in general not familiar with Markov reward models, so they are not inclined to use this formalism to model their system;
(2) Markov reward models are very "low-level" from a system designer's point of view, and thus the modelling process will be very error-prone;
(3) models tend to become too big to handle manually, i.e. the model specification becomes too complex.

Berson et al. [7] distinguish two representations of a model. On the one hand there is what they call the *analytical representation* of a model. This is the representation that is directly suitable for a numerical evaluation. In the performability modelling context an analytical representation of a model would be a Markov reward model and the numerical evaluation would use one of the techniques discussed in Section 2. On the other hand there is the *modellers representation* of a model. This is a description in a symbolic form oriented towards the specific application (i.e. the system to be modelled). Clearly, most system designers prefer to use the modellers representation rather than the analytical representation.

In the performance modelling context a similar distinction applies. Beilner claims that a performance model specification used in a performance modelling tool should be independent of any particular performance evaluation technique [4]. This is in line with the statement above that for system designers the analytical representation of a model, which is typically tailored to some specific evaluation technique, is hard to use. Model descriptions should therefore be based on a formalism close to the application domain and as independent as possible from underlying evaluation techniques.

Let us now define a general framework for performability modelling tools, called the *General Modelling Tool Framework* (GMTF) originally proposed in [35]. Note that although we introduce the GMTF here in a performability modelling context, it is widely applicable in the context of quantitative systems modelling. For instance, in [40,84], tools for performance analysis of communication systems based matrix geometric methods and Markovian analysis were presented that have a structure conforming to the GMTF. A similar layering structure for model-based evaluations has recently also been proposed by Lepold [51] as a generalization of the open layered architecture for dependability analysis as presented in [62].

We introduce a hierarchy of modelling formalisms ranging from $\mathcal{F}_0$ (the lowest level) to $\mathcal{F}_n$ (the highest level). $\mathcal{F}_0$ yields models that are directly suitable for numerical evaluation whereas $\mathcal{F}_n$ is the formalism closest to the application domain. We define *$\mathcal{F}_i$-modelling* as the process of abstracting, simplifying and/or rewriting a system description $\mathcal{S}$ in such a way that it fits some formalism $\mathcal{F}_i$. The result of this process is called an $\mathcal{F}_i$-model $\mathcal{M}_i$ of $\mathcal{S}$. An $\mathcal{F}_i$-model $\mathcal{M}_i$ of $\mathcal{S}$ can be modelled in another formalism $\mathcal{F}_{i-1}$, provided $i \geq 1$, yielding an $\mathcal{F}_{i-1}$-model $\mathcal{M}_{i-1}$ of $\mathcal{S}$. This is called $\mathcal{F}_{i-1}$-modelling. The lowest level formalism is $\mathcal{F}_0$ which coincides with the analytical representation as discussed by Berson et al. [7]. When $\mathcal{F}_i$ is the highest level formalism, most of the user activity in the modelling process will be $\mathcal{F}_i$-modelling. The lower level modelling activities will often be partially or completely automated.

Once we have a model it can be evaluated. The evaluation of an $\mathcal{F}_0$-model $\mathcal{M}_0$ yields results in the formalism $\mathcal{R}_0$. For this evaluation numerical techniques like those indicated in Section 2 can directly be applied since the formalism $\mathcal{F}_0$ has been chosen to directly suit those numerical evaluation techniques.

The evaluation of an $\mathcal{F}_0$-model $\mathcal{M}_0$ is called a $\mathcal{V}_0$-*evaluation*. The results presented in the formalism or domain $\mathcal{R}_i$ ($i \geq 0$) can be further processed or enhanced to some higher level $\mathcal{R}_{i+1}$. This is called an $\mathcal{E}_{i+1}$-*enhancement*. Often these enhancements can be done automatically.

When we have an $\mathcal{F}_j$-model $\mathcal{M}_j$ of some system $\mathcal{S}$ we generally want to evaluate this model and to obtain measures in a formalism of the same level. We denote the level of this formalism as $\mathcal{R}_j$. We can also say that the results are given in *domain* $\mathcal{R}_j$. We define a *virtual evaluation* $\mathcal{V}_j$ as the process of subsequently modelling $\mathcal{M}_i$ ($1 \leq i \leq j$) in formalism $\mathcal{F}_{i-1}$, until an $\mathcal{F}_0$-model $\mathcal{M}_0$ is obtained, followed by the $\mathcal{V}_0$-evaluation and the subsequent enhancements $\mathcal{E}_1$ through $\mathcal{E}_j$. Schematically, we have the structure as depicted in Fig. 1. This structure represents the GMTF. The small boxes represent system models (right-hand side) or evaluation results (left-hand side). The large box represents the actual mathematical evaluation. In the context of performability modelling this will often be a Markov reward analyzer. The single pointed arrows represent automatic translations of one formalism into another. The double pointed arrows represent the virtual evaluations.

Hierarchical modelling also fits in the GMTF. In hierarchical model description formalisms, high-level models are described in terms of lower-level models and interactions between them. This intuitive "describing in terms of" conforms to the above mentioned $\mathcal{F}_{i-1}$-modelling of $\mathcal{M}_i$.

Although the GMTF can guide thinking about and development of tools, practice often tends to deviate from frameworks or reference models. As we will see, some tools allow their users to partially model their

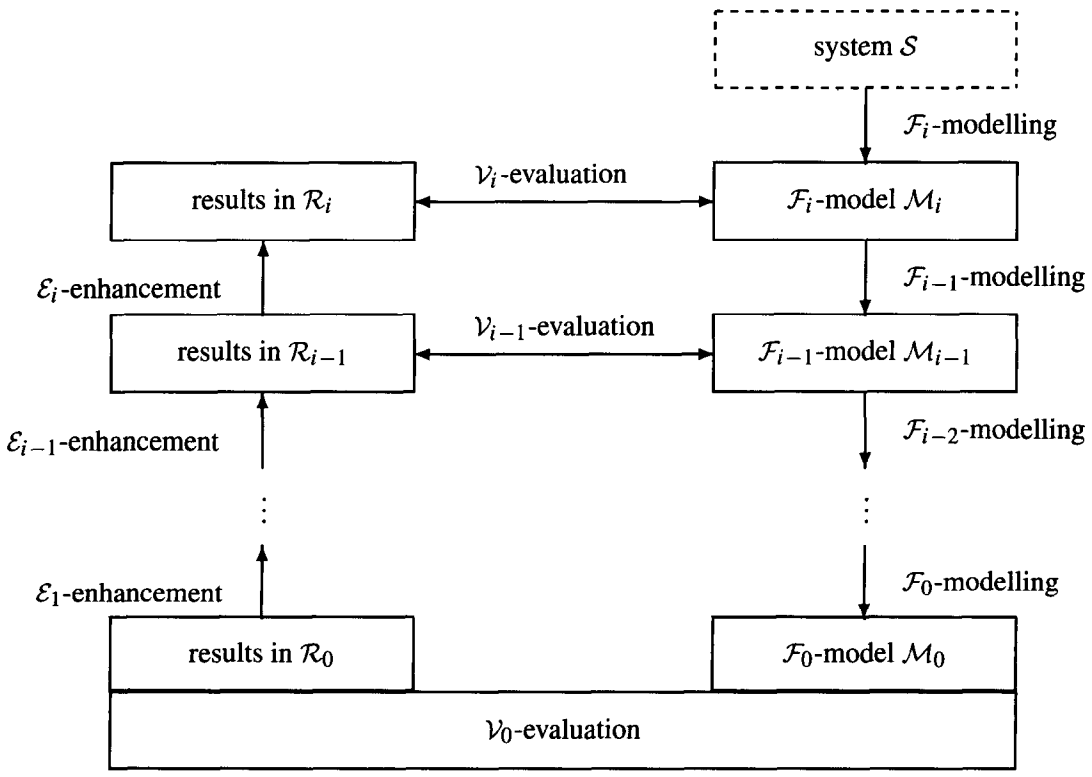


Fig. 1. General modelling tool framework.

system in formalism $\mathcal{F}_1$ and partially in formalism $\mathcal{F}_0$. Sometimes the results are presented in domain $\mathcal{R}_i$ whereas the model is made in formalism $\mathcal{F}_j$ with $i < j$. In these cases one cannot speak of $\mathcal{V}_j$-evaluations.

Important to note is the fact that the mentioned levels are not absolute. A tool allowing for $\mathcal{V}_1$-evaluations can from an application point of view be "a higher level" tool than another tool which allows for $\mathcal{V}_2$-evaluations. To make this more concrete, consider the object-oriented modelling tool Tangram [7,88]. As we will see later, this tool uses communicating objects for the description of Markov models. The communicating objects can be considered as models in the formalism $\mathcal{F}_1$. Tangram can be extended to allow for a modelling activity similar to the modelling activity that is required when using Save. Consequently, a possible $\mathcal{V}_2$-evaluation of Tangram corresponds to the $\mathcal{V}_1$-evaluations of Save, i.e. the $\mathcal{V}_2$-evaluations of Tangram are *not* of higher level than the $\mathcal{V}_1$-evaluations of Save.

Regarding the transparency of the $\mathcal{V}_0$-evaluations, the following remarks can be made. In general, we can say that the $\mathcal{F}_0$-models are descriptions of Markov reward models. It can be discussed whether the choice of an algorithm in the $\mathcal{V}_0$-evaluation should be made internally, or whether these choices should be part of the $\mathcal{F}_0$-models (as a kind of directives). Clearly, it is very difficult to totally separate "low level details" from the "real modelling work". However, at least an intuitively appealing attempt in this direction should be made. Thus, we feel that we must be able to do higher-level evaluations without detailed knowledge of the underlying techniques, however, if we do know the underlying techniques, we should be able to control them.

As a conclusion of the presentation of the GMTF we would like to put forward a number of criteria related to the GMTF which should be fulfilled by a well-designed tool:

balancedness: the modelling formalism and the output formalism should be of the same abstraction level, i.e. the results should be presented in the domain $\mathcal{R}_i$ whenever the modelling is done with formalism $\mathcal{F}_i$;

extensibility: the tool should most optimally be extendible towards higher-level evaluations, i.e. given that the tool allows $\mathcal{V}_i$-evaluations, it should be possible to adapt the tool in such a way that it provides $\mathcal{V}_{i+1}$-evaluations;

transparency: the $\mathcal{V}_0$-evaluation should be as transparent as possible.

4. The performability modelling tool survey

In this section we survey twelve performability modelling tools. From Section 2 we know that performability modelling incorporates both performance analysis (for obtaining the rewards) and Markov reward model analysis. A performability modelling tool should therefore include both aspects. As we will see this is not yet commonplace. The tools that we survey can, however, all in some way be used for some aspects of performability modelling. In our choice, we have restricted ourselves to Markov reward model-based tools. We do not include general simulation-based tools. Also, we do not include tools purely aiming at dependability (e.g. Figaro [8], Surf [18]) or tools mainly aiming at performance (e.g. GreatSPN [12], DSPNexpress [52]) or tools aiming at system optimization (e.g. Penelope [54]) although we appreciate the contributions of these tools as such.

The following tools have been included in the survey (in order of their first publication): Metaphor, Numas, Metasan, Metfac, Save, Sharpe, SPNP, Tangram, Penpet, UltraSAN, Surf-2 and DyQNtool$^+$. For each tool we:

(1) provide references and implementation details;
(2) discuss the model class and the evaluation techniques;
(3) discuss the input and possible output of the tool;
(4) discuss how the tool fits in the GMTF;
(5) summarize and evaluate its most important characteristics.

In Table 1, we present for each tool the models and measures supported, the numerical techniques employed, the required input and output format, the format of the rewards, its fitting in the GMTF, the language it has been implemented in, the platforms it operates on, as well as references.

It is difficult to report in general terms on the performance of the tools. The tool performance very much depends on the models one studies, i.e. their size, structure and the numerical parameters involved, the measures one is interested in, i.e. only steady-state measures or also distributions, and the actual encoding of the algorithms. For a comparison of the complexity of the numerical techniques, we refer to the relevant literature mentioned in Section 2. How fast the various high-level model descriptions can be translated to the lower-level Markov reward models, and how efficient the implemented solution techniques are, remain interesting questions. To shed light on them, a number of cases should be tackled by all the tools and the resulting performances be compared. This, however, goes beyond the scope of the current paper.

Table 1
Overview of performability modelling tools

	Metaphor	Numas	Metasan	Metfac
first published	1984	1985	1986	1986
model class	acyclic semi-Markov non-increasing rewards	Markov reward	Markov reward impulse and rate rewards	Markov reward
measures	PDF	SS	SS, TP, CP, PDF	SS, TP, CP, PDF
techniques	state trajectory enumeration, explicit integration	Gaussian elimination Gauss–Seidel iterative aggregation	Gaussian elimination Gauss–Seidel uniformization simulation	Gaussian elimination direct integration
rewards computed	off-line	from QN model	C function	off-line
rewards inserted	state level	automatically	SAN level	state level
input	interactive	interactive graphical Hit interface	Sanscript language	production rule system
output	tables	tables	tables	tables
GMTF	$\mathcal{F}_0$ input $\mathcal{R}_0$ output $\mathcal{V}_0$ evaluations	$\mathcal{F}_1$ input $\mathcal{R}_1$ output $\mathcal{V}_1$ evaluations	$\mathcal{F}_1$ input $\mathcal{R}_i$ output $\mathcal{V}_i$ evaluations	$\mathcal{F}_i, \mathcal{F}_1$ input $\mathcal{R}_0$ output $\mathcal{V}_{0,1}$ evaluations
language	APL and C	Simula 67, C	C	Fortran 77
system	UNIX systems	Siemens BS2000 SUN 4, APOLLO	SUN 3, VAX	VAX (VMS)
literature	[23,24]	[60]	[57,78–81]	[9,10]

Table 1 (*continued*)

	Save	Sharpe	SPNP	Tangram
first published	1986	1986	1989	1989
model class	Markov reward	(semi-)Markov reward	Markov reward	Markov reward
measures	SSP, TP, CP, PDF sensitivities	SSP, TP, CP, PDF	SSP, TP, CP, MRTA sensitivities	state probabilities
techniques	SOR, Gauss–Seidel randomization importance sampling	SOR, Gauss–Seidel randomization symbolic algorithms	SOR, Gauss–Seidel randomization	—
rewards computed	off-line	off-line, in Sharpe	C function	off-line
rewards inserted	state level	state level	SRN level	state level
input	Save input language	MRM enumeration	CSPL	communicating objects
output	tables	tables	tables	Query language
GMTF	$\mathcal{F}_0$, $\mathcal{F}_1$ input $\mathcal{R}_1$ output $\mathcal{V}_{0,1}$ evaluations	$\mathcal{F}_0$ input $\mathcal{R}_0$ output $\mathcal{V}_{0,1}$ evaluations	$\mathcal{F}_1$ input $\mathcal{R}_1$ output $\mathcal{V}_1$ evaluations	$\mathcal{F}_i$ input $\mathcal{R}_i$ output $\mathcal{V}_{0,i}$ evaluations ($i \geq 1$)
language	Fortran 77	C	C	OO-Prolog
system	IBM/370	Sun 4,VAX(VMS)	SUN 4, VAX(VMS), Convex NeXT, RS/6000, OS/2	SUN 3
literature	[25–29,41]	[73–77]	[15,16]	[7,66]

	Penpet	UltraSAN	DyQNtool$^+$	Surf-2
first published	1991	1991	1992/5	1993
model class	Markov reward	Markov reward impulse and rate rewards	Markov reward	Markov reward
measures	SSP, TP	SSP, TP, CP, PDF	SSP, TP, CP, MRTA	SSP, TP, MRTA
techniques	Gaussian elim., Gauss–Seidel, SOR ACE, randomization	LU-decomposition, SOR, randomization importance sampling	Gauss–Seidel, SOR randomization	—
rewards computed	from SPN model	C function	from PFQN model	function
rewards inserted	automatically	SAN level	automatically	SPN/state level
input	textual	graphical	textual, C-based	graphical
output	tables, graphs	tables	tables	tables, graphs
GMTF	$\mathcal{F}_{1,2}$ input $\mathcal{R}_1$ output $\mathcal{V}_1$ evaluations	$\mathcal{F}_i$ input $\mathcal{R}_i$ output $\mathcal{V}_i$ evaluations ($i \geq 1$)	$\mathcal{F}_1$ input $\mathcal{R}_1$ output $\mathcal{V}_1$ evaluations	$\mathcal{F}_{0,1}$ input $\mathcal{R}_{0,1}$ output $\mathcal{V}_{0,1}$ evaluations
language	C	C, C++	C	C, ADA
system	UNIX systems	SUN 4, Decstation RS/6000, Convex	SUN 4	SUN 4
literature	[45,49–51]	[20,65,82,83]	[34–36,39]	[6]

4.1. Metaphor

The tool Metaphor (Michigan EvaluaTion Aid for PerpHORmability) has been developed by Furchtgott and Meyer at the University of Michigan [23,24]. An APL as well as a C/UNIX version of Metaphor exists.

Model class and evaluation techniques. Metaphor addresses only models of systems that are non-repairable and non-recoverable. Having non-repairable models implies that the stochastic process $X = (X_t, t \geq 0)$ is acyclic; non-recoverability means that whenever the process structure allows a transition from state i to state j then $r(j) \leq r(i)$, i.e. the system is not becoming better in time. On the other hand, the stochastic process X may be semi-Markovian. Since the stochastic processes are acyclic, one can enumerate all state trajectories from the unique starting state to all the down states. Knowing the state transition behaviour, the probabilities of occurrence of a particular state trajectory as well as the distribution of the cumulative performance of the system over that state trajectory can be computed. Combining these two yields the performability distribution $F(t, y)$. In Metaphor, techniques have been implemented to reduce the number of state trajectories. No explicit mention is made of the maximum size of models Metaphor can cope with. However, an example with hundreds of state trajectories is discussed by Furchtgott [24].

Model input and output. The models are input interactively. The rewards for all possible structure states have to be input separately. This implies that a separate performance modelling activity is necessary. The output of the tool gives values for the function $F(t, y)$ in tabular form.

GMTF fitting. Metaphor is an $\mathcal{F}_0$-modelling tool. The output is in the $\mathcal{R}_0$-domain. The tool can be used for $\mathcal{V}_0$-evaluations.

Evaluation. Metaphor was the first tool for performability modelling. Its model class is restricted in the sense that only acyclic (non-repairable), non-recoverable stochastic processes can be used. There are, however, no restrictions on the state residence distributions. As models become large, the enumeration of all the state trajectories and the integration over them can be a severe drawback in using the tool. Another disadvantage of the tool is the fact that the possibly large number of rewards has to be calculated off-line. Both the input and the output of the tool are textual.

4.2. Numas

Numas (NUmerical Methods for the Analysis of computer Systems) has been developed by Müller-Clostermann at the University of Dortmund [60]. Numas is basically a performance analysis tool, however, it is possible to extend each queueing station with fault-tolerance characteristics, so that performability measures can be obtained. Numas has been implemented in Simula 67 for Siemens BS2000 systems. The algorithms used in Numas are also accessible from the hierarchical modelling tool Hit as discussed by Beilner et al. [5]. For Hit, also versions exist for IBM, APOLLO and SUN systems. A X11-based graphical interface is also available.

Model class and evaluation techniques. With Numas queueing network models are solved by numerically computing the steady-state distribution of the underlying Markov chain. Within every multi-server queue individual servers may fail with load dependent rates and can be repaired with a rate dependent on the number of servers that is still available, independently of all other queueing stations. Thus, the state of the network consists of a part describing the distribution of customers over the queueing stations and of a part describing the so-called degradation mode of each queueing station. Groups of states with the same degradation mode are said to belong to one macro state. By the fact that there are normally large time-scale differences between transitions inside and between macro states, it is possible to decompose the overall

model. For every macro state, a Markovian submodel for the associated performance model is identified and analyzed. The submodel results are used in a numerical analysis of the Markov chain describing the transitions between degradation modes.

For obtaining the steady-state probability vectors iterative and direct methods are used such as Gaussian elimination, Gauss–Seidel iterations and iterative aggregation. Models with thousands of states can easily be handled.

Model input and output. The methods in Numas can be accessed via two user interfaces. Both user interfaces allow the user to input the so-called degradable queueing network models at a high level. The associated Markov model is generated automatically. There is an interactive Numas interface available in the Siemens/Simula implementation. The Numas methods can also be accessed via the hierarchical modelling tool Hit. The output of the tool is given in tabular form and is related to the original modelling constructs.

GMTF fitting. Numas input takes place via a specialized interactive dialogue and its output is given in terms of the input modelling elements. The user is not aware of the underlying Markov reward models. Consequently, Numas allows $\mathcal{V}_1$-evaluations.

When using the Hit system and HI-SLANG (Hit System Language) as input language, there are very nice features for hierarchical modelling. The outputs can then be tailored to the highest level of the model description. With Hit, we thus have $\mathcal{V}_i$-evaluation capabilities ($i \geq 1$).

Evaluation. For steady-state performability analysis, Numas provides powerful constructs. All necessary information is computed by the tool. A limitation of the tool is that it is not possible to model dependencies in the failure and repair processes of the various queueing stations. Also, only steady-state measures are computed.

4.3. Metasan

The tool Metasan (Michigan Evaluation Tool for the Analysis of Stochastic Activity Networks) has been developed by Sanders et al. [57,78–81] at ITI (Industrial Technology Institute) in cooperation with the University of Michigan. It has been written in C with the help of the tools Yacc and Lex. Versions for SUN 3 and VAX (both UNIX) exist.

Model class and evaluation techniques. The tool Metasan is based on the Stochastic Activity Network (SAN) approach of describing complex systems [57]. SANs consist of places and activities (transitions in GSPN terminology). Activities can be either stochastically timed or instantaneous. In the timed case, general distributions are allowed, however, if non-exponential distributions are used, simulation should be used to solve the model. So-called cases can be associated with each activity. Cases are generalizations of probabilistic switches [3]. Upon completion of an activity, one of the cases is chosen probabilistically. Both the timing distributions and the case probabilities can be marking dependent. Input and output gates that connect places to activities and vice versa determine the flow of tokens through SAN. Input gates have a predicate that specifies whether the activity is enabled or not, and a function that specifies how the tokens from the input places are redistributed over the output places of the activity. Output gates only have the latter function; they do not have predicates.

With Metasan, it is possible to construct models in a hierarchical fashion using a macro facility, i.e. submodels, SANs in itself, can be defined which are subsequently used (one or more times) in higher-level models. The overall SAN model describes the structure-state process. In terms of place occupancies, the user can associate rewards with all states. From these, an underlying Markov reward model is derived. The underlying Markov model is solved for the steady-state performability measures using either direct

or iterative methods. Transient performability measures are obtained using uniformization. A number of algorithms have been implemented for obtaining $F(t, y)$ under various model restrictions. A specialized algorithm is incorporated for obtaining $E[Y(t)]$.

It is also possible to use simulation as an evaluation technique. Steady-state as well as transient measures (terminating simulation) may be obtained in this way. Confidence intervals are also obtained and can be used to specify stop criteria for the simulation, e.g. stop if 95% confidence interval width is within 10% of the estimate.

Metasan is one of the two performability modelling tools (the other one is UltraSAN (see Section 4.10)) that combines rate-based and impulse-based Markov models.

Model input and output. Model input and output are textual. The input consists of two parts: a part describing the actual model and a part describing the experiment, i.e. the measures to be obtained, the techniques to be used, etc. The model description part is done with the SAN description language Sanscript and is independent of the experiment description. The output is given in tables.

GMTF fitting. With the Metasan modelling language, the modelling activities take place at a higher level than at the Markov reward level, i.e. $\mathcal{F}_1$-modelling. Due to the macro facilities, a form of higher-level modelling is also possible. Rewards are computed as a general C function of the SAN marking and/or the activity completions. As such, the rewards are not derived from a performance model, albeit that the C functions can be an "implementation" of some queueing model. The output is given in a form that can be understood at the level of model description, i.e. in the $\mathcal{R}_i$-domain ($i \geq 1$). Thus, $\mathcal{V}_i$-evaluations ($i \geq 1$) can be done with the tool.

Evaluation. Metasan provides a very general framework for building performability models. The tool supports analytical as well as simulative evaluation techniques. The separation between a model description and a model evaluation part is a nice feature since it allows the modeller to evaluate one particular model with various evaluation techniques. Input of the tool is textual. The output is in the form of tables. Rewards have to be computed separately.

4.4. Metfac

The tool Metfac (Modelación y Evaluación de la Tolerancia a Fallos Asistida por Computador) has been developed by Carrasco and Figueras at the Polytechnical University of Catalunya [9,10]. It has been written in Fortran 77 and runs on VAX (VMS) systems.

Model class and evaluation techniques. The Metfac system allows the analysis of Markov reward models. The models are specified by means of a production rule system. In short, the idea behind this is the following. The system structure is described by a number of state variables. For all the possible values (value ranges) of the state variables it is specified which events can occur, the rate with which they occur and how they affect the state variables. In this way, all possible finite Markov chains can be described. Steady-state as well as transient performability measures can be obtained. A measure close to the cumulative performability measure can also be obtained: the so-called serviceability $S(y)$, i.e. the probability distribution that an amount of work y has been done before the first failure occurred. For obtaining steady-state measures direct methods are employed. The transient measures are obtained using numerical integration procedures especially suited for stiff problems. Large state spaces are dealt with by an approximate technique called dissolving. Dissolving basically means that states with very low probabilities are omitted but accounted for by altering transition rates in their predecessor states. The evaluation routines in Metfac are especially

developed for large, sparse Markov models. It is suggested that the tool is able to deal with models having thousands of states.

Model input and output. The production rule system has to be input textually. The rewards have to be input separately for all possible states. The translation from the production rule system to the underlying Markov reward model is done automatically. The output is given in tabular form.

GMTF fitting. Modelling systems with the production rule mechanism of Metfac typically is an $\mathcal{F}_1$-modelling activity. The specification of the rewards to be associated with every global state is a lower-level model activity, i.e. $\mathcal{F}_0$-modelling. The output of the tool is in the $\mathcal{R}_0$-domain.

Evaluation. Metfac provides a very powerful mechanism for constructing the dependability part of a Markov reward model that can subsequently be used for performability analysis. Rewards have to be supplied manually. A large variety but not all of the performability measures can be obtained. In the implementation of the tool, much emphasis has been put on state dissolving techniques and on the exploitation of the sparsity of the Markov generator in the evaluation routines.

4.5. Save

Save (System AVailability Estimator) has been developed by Goyal et al. [25,28,29] at IBM Yorktown Heights in cooperation with Duke University. It has been written in Fortran 77 and runs on an IBM System/370. Originally designed for modelling ultra-dependable computer systems, it has been extended for performability modelling.

Model class and evaluation techniques. Save allows for Markovian structure state processes. With every possible structure state a performance level can be associated. Steady-state measures are obtained using the iterative technique of successive over-relaxation (SOR). Transient measures are obtained using the randomization approach. Various specialized algorithms have been implemented to compute the $F(t, y)$. A special feature is the possibility to obtain the sensitivity of the results with respect to model parameters [27,41]. Simulating the Markovian models, which is imperative when the models become very large, is also possible. In that case an importance sampling technique is used, which greatly reduces the required simulation time [30]. As far as the Markov approach is concerned, models with hundreds of thousands of states can be handled.

Model input and output. The user can choose the Markov input approach or the numerical input approach. In the former, the user has to specify all the Markov states, their corresponding rewards and their outgoing transitions. In the numerical approach the user can specify the models at a higher level, using powerful basic modelling constructs. In both cases, the rewards have to be supplied by the user. Given a model input, Save automatically constructs the corresponding Markov reward model after which the user can interactively indicate which measures have to be computed. All input is textual. All output is given in tabular form.

GMTF fitting. The Save input language is especially developed for modelling the dependability of systems, and thus supports $\mathcal{F}_1$-modelling activities. The input of the performance aspects, i.e. the rewards, is a typical $\mathcal{F}_0$-modelling activity. The output is given in terms of the modelling constructs that are used when specifying the dependability aspects.

Evaluation. Save is a powerful tool for the analysis of Markov reward models. Its constructs for specifying the structure state process are very powerful and so are the implemented evaluation techniques. Steady-state, transient and cumulative performability measures can be computed by the package.

4.6. Sharpe

Sharpe (Symbolic Hierarchical Automated Reliability/Performance Evaluator) has been developed by Sahner et al. [73–77]. Sharpe allows for various types of modelling and for various types of evaluation techniques. In this section we will only discuss the capabilities of Sharpe as far as performability is concerned. Sharpe has been implemented in C and runs on SUN (UNIX) and VAX (VMS) systems.

Model class and evaluation techniques. Sharpe allows for the construction and analysis of Markov and semi-Markov chains. The Markov chains must either be acyclic, irreducible (every state is reachable from every other state) or phase-type (there is at least one absorbing state and every non-absorbing state is transient). The semi-Markov chains must be acyclic or irreducible. With every state a reward can be associated.

Assuming the system to be down in the absorbing states and operational in the transient states, the phase-type Markov chains can be used to calculate the system reliability, the mean time to failure, and the time to failure distribution symbolically. By using convenient rewards also measures like mean computation to failure (originally defined in [2]) and the distribution of the cumulative reward upon absorption, the expected reward rate at time t and the expected cumulative reward at time t can be obtained.

When the (semi-)Markov chain is irreducible, the steady-state performability can be calculated. When acyclic semi-Markov chains are used, not only exponential distributions but also polynomial exponential distributions ("exponomials") can be used, i.e. distributions of the form $F(t) = \sum_i a_i t^{k_i} e^{b_i t}$.

The steady-state results are obtained with either Gauss–Seidel iterations or by using SOR. Transient quantities are derived in semi-symbolic form [73–77,91] or in numeric form by using randomization [31,32,87,88].

Model input and output. The specification of the Markov models in Sharpe is done by enumerating all the state transitions and the rewards associated with them. A nice feature of Sharpe is that it is possible to model hierarchically, also between different types of models. Thus, it is possible to calculate the rewards using a queueing network analysis (also provided by Sharpe) and combine them with a Markov reward model (see the approach used in DyQNtool$^+$).

GMTF fitting. The standard Sharpe tool environment allows for $\mathcal{F}_0$-modelling. By clever usage of the hierarchical modelling capabilities of Sharpe it is possible to enhance this level.

Evaluation. The real strength of Sharpe lies not in pure performability modelling, but more in its capability to use various modelling techniques in a combined fashion. The input of the tool seems to be slightly inconvenient for performability models (Markov reward models), especially for larger ones. On the other hand, the availability of symbolic evaluation techniques for a subclass of problems is attractive.

4.7. SPNP

The Stochastic Petri Net Package has been developed by Ciardo et al. at Duke University [15,16]. SPNP is a very general and flexible package for the construction of stochastic Petri nets (SPNs) and the subsequent analysis of the underlying Markov reward models (the class of SPNs supported by SPNP is sometimes referred to as stochastic reward nets (SRNs)). SPNP has been implemented in C and runs on SUN , CONVEX and NeXT systems (all UNIX), RS/6000 (AIX), PS/2 (OS/2) and VAX (VMS).

Model class and evaluation techniques. With SPNP all types of Markov reward models can be constructed. There are no limitations on the rewards. The Markov reward models are derived from a SPN description. The rewards can be specified as normal C functions over the number of tokens in the places.

This allows for a very flexible reward structure construction, since the used C functions can invoke other functions, e.g. to do some performance analysis.

The resulting Markov reward model can be analyzed for steady-state measures, using either Gauss–Seidel iterations or SOR. For Markov reward models with absorbing states, the obtained cumulative reward until absorption can be calculated. Transient and cumulative measures are obtained using randomization. Sensitivities can be obtained for steady-state measures, again using Gauss–Seidel or SOR techniques, as well as for transient measures, thereby using randomization.

Model input and output. The input language for SPNP is CSPL, i.e. the C-based SPN Language. A CSPL description is a normal C file; it is compiled with the C compiler and linked with a number of other C programs which together constitute the SPNP. The output of the tool takes place via C functions. This allows the tool user to tailor the output to his specific needs.

GMTF fitting. SPNP can be used for $\mathcal{V}_1$-evaluations. The possibilities of the tool to allow for the use of parametrized subnet specifications indicates that the modelling and evaluation can be enhanced to higher levels than the standard $\mathcal{F}_1$-modelling and $\mathcal{V}_1$-evaluations. Rewards are input at the SPN level as a general C function over the markings. As with Metasan, this derivation is not model based.

Evaluation. SPNP is a powerful tool for the construction of Markov reward models. Since the model description is done via standard C, the full power of the C programming language is available. This makes the tool very flexible and easy to use. This as well as the fact that rewards can be associated with every marking by using normal C functions opens up numerous possibilities to further facilitate the modelling process. A wide variety of performability measures can be obtained including steady-state, transient and cumulative measures.

4.8. Tangram

Tangram is a general object-oriented tool environment developed by Berson et al. at the University of California at Los Angeles [7,66,88]. Tangram provides a layered tool environment which can very easily be tailored to specific application domains, given that evaluation techniques are available. Tangram has been implemented in object-oriented Prolog and C for SUN 3 systems running UNIX.

Model class and evaluation techniques. Tangram is a very general modelling environment which can currently be used for performance analysis, Markov reward analysis and reliability analysis. Which methods are implemented for the Markov reward analysis remains unclear from the cited papers.

The strong points of Tangram are its extremely flexible input and output format. Models are specified as collections of objects. Objects are parametrized instances of object types. Every object type has an internal state which changes upon the completion of internal events. Internal events can also cause messages to be sent to other object types. Reception of messages by an object type also causes events within the receiving object type. The overall model state is the collection of the internal states of all the objects. With every state a reward can be associated. Since object types can inherit properties from parent object types, complex models can be constructed in a stepwise fashion. In this way it is possible to define a set of high-level objects specially tailored for a particular application. Berson et al. indicate that they have constructed a set of high-level object types that allows Tangram users to use the same modelling constructs as are provided by the Save interface.

Model input and output. The input of Tangram is completely textual, program like. The output of Tangram takes place via a query language. Queries can be defined over objects, their internal state and their rewards.

GMTF fitting. The Tangram tool environment allows for the easy construction of Markov reward models. Models can be built hierarchically, so we typically have to do with $\mathcal{F}_i$-modelling ($i \geq 1$). The output can be enhanced to the same level as the input by using the query language. Tangram therefore typically allows for $\mathcal{V}_i$-evaluations ($i \geq 1$). Rewards, however, have to be calculated separately.

Evaluation. Tangram is a very flexible modelling environment which greatly eases the construction of Markov reward models. It allows for hierarchical modelling and has a powerful query mechanism for obtaining measures that are complex to specify.

4.9. Penpet

The tool Penpet (PEtri Net based Performability Evaluation Tool) has been developed by Lepold at Siemens AG, in cooperation with the University of Mulhouse [49–51]. The system has been built with the C programming language under the UNIX operating system, making use of X-windows and Motif.

Model class and evaluation techniques. With Penpet the dependability model part is at the highest level described by a structure formula. Models (so-called macromolecules) are defined that consist of a replication of a number of submodel parts (molecular clusters) and/or components (atoms). This definition is done by means of a so-called system structure formula, a textual description that closely resembles the description of molecules in chemistry. For each of these submodel parts and components, it is specified how many there are, how many need to be operational for the next hierarchically higher model to be operational, etc. Failure rates and coverage factors are also specified per component and/or per submodel.

Components and submodels used in this way are described themselves by one of a number of library-provided GSPNs. Extensions to this library of standard submodels can also be made. The class of GSPNs is the so-called Performability Adapted SPNs (PASPNs), which fall in the class of GSPN models that can be analyzed with Tompsin, a package also developed at Siemens [48]. A nice feature of Tompsin is that it allows for approximate hierarchical modelling, which decreases the size of the state space of the performance models [45].

From the system structure formula an overall GSPN describing the dependability aspects is derived. From this GSPN, all the possible system structure states are derived as well as the Markov chain underlying the GSPN. From all the system structure states, via a set of functions (as in the case of dynamic queueing networks, see DyQNtool$^+$ below) the parameters of the performance models are derived. These parameters might include initial markings, firing rates, etc. A reverse dependence is also possible, e.g. failure rates in the structure state model might be dependent on the rewards that are derived. Once the performance models have been constructed they are evaluated by normal GSPN analyses, using the package Tompsin. Combining the structure state process with the obtained rewards, a Markov reward model is obtained.

For deriving steady-state performability measures such algorithms as Gaussian elimination, Gauss–Seidel iterations, SOR and the LSQR algorithm (an extension of the conjugate gradient method) have been implemented. For deriving point performability measures the ACE algorithm [53] has been implemented as well a randomization procedure.

Model input and output. The input to Penpet is textual. The user can textually input the system structure formula as well as the needed GSPN submodels, if they are not readily available from the library. The output of the tool is in the form of tables, or in the form of graphs, thereby making use of the Xgraf package.

GMTF fitting. The tool Penpet allows for the construction of Markov reward models by techniques that are specially designed for the performability analysis of fault-tolerant multiprocessor systems. Also the rewards are derived automatically from high-level SPN model, therefore $\mathcal{F}_1$-modelling is possible. The

output is presented in the $\mathcal{R}_1$-domain, so Penpet is capable of $\mathcal{V}_1$-evaluations. The structure state formulae can be seen as $\mathcal{F}_2$-models since they are higher-level descriptions of GSPNs.

Evaluation. Penpet allows for the construction of Markov reward models by techniques that are especially designed for the performability analysis of fault-tolerant multiprocessor systems. The way of describing higher-level models as clusters of lower-level models that can be specified separately is very flexible. Also the mutual dependence between performance models and structure state models is a very nice feature.

4.10. UltraSAN

The tool UltraSAN has been designed by Sanders et al. [20] at the University of Arizona, as a successor to Metasan. Recently, a second release has been made available [83]. UltraSAN has completely been written in C++ (user interface) and C, making use of the X-windows environment, and runs on SUN 4, DEC Decstation, RS/6000 and Convex systems.

Model class and evaluation techniques. UltraSAN is, like Metasan, based on stochastic activity networks. Models are input using a graphical interface. It is possible to construct models in a hierarchical fashion. Submodels can be defined which are subsequently used (one or more times) in higher-level models by use of the so-called *replication*-operator. Submodels of various kinds can be combined in a higher-level model by means of the *join*-operator. The joining and replicating of submodels can be done iteratively, thus providing real hierarchical modelling. Apart from exponentially timed activities, also deterministically timed transitions are allowed, albeit in a restricted fashion [52].

Steady-state measures are solved either direct or iteratively. In the former case a variant of the well-known LU-decomposition algorithm is used which reduces fill-in by a heuristic technique for pivot selection and which deals in a special way with very small elements. In the latter case a method based on successive over-relaxation is used. For transient and cumulative measures a randomization method is used. It yields transient state probabilities, expected values of cumulative sums as well as probability distributions. For the distribution of the accumulated reward during a fixed time interval, a randomization procedure has been developed that utilizes so-called path truncation to keep memory and CPU requirements small [68]. Also, for all the above measures, rate and impulse rewards can be combined.

For all the analytical and numerical techniques counts that they use what is called a *reduced base model* for their analysis. It hereby makes use of the structure of the model, in terms of submodels, joins and replications. In this way, state space reductions of several orders of magnitude can be achieved [82], thus allowing for real applications to be modelled and solved numerically. The so-called stochastic well-formed nets (SWFNs) [13] can be regarded as a special case of the hierarchical modelling possibilities provided by UltraSAN. SWFNs can be seen as a single SAN which is replicated. Also, for SWFNs, no tool support is currently available. UltraSAN allows for multiple levels of such replications and also is able to combine different models (the join operator).

UltraSAN also provides simulation as a method of evaluation. Both steady-state simulations and transient simulations are possible. By making use of the structure of the SAN model to be analyzed, very efficient simulation techniques have been implemented, in which a single event-list is exchanged for an event-tree [83]. This event-tree follows the tree-form of the model itself in terms of submodels that are iteratively joined and replicated. For large models, this structuring of the event-list increases the efficiency of the simulation. An importance sampling technique is used to speed up the simulations [65].

Model input and output. UltraSAN makes use of a graphical interface. The SAN submodels are input with the SAN editor. The timings associated with the activities are input via menus. The activity rates and

output gate functions are described in C. Once the submodels are defined they can be composed into an overall model with the graphical composed-model editor. Subeditors for the join and replicate functions are provided. The performability measures of interest are specified using a measure editor.

Once a model is totally specified UltraSAN generates a solution program which upon execution yields the desired measures in tabular form. The generated program can be a simulation program or a program for any of the included numerical techniques.

Instead of generating a single executable which, upon execution, solves the model, also a multiple run option is provided. This allows the modeller to define series of experiments by defining global variables and indicating their value-ranges. In the SAN definitions, references to these variables can be made. For every possible combination of parameter values, UltraSAN generates the corresponding solution program. These solution programs are then executed on a cluster of workstations in parallel. This allows for the easy and fast execution of parametric studies.

GMTF fitting. With UltraSAN the modelling activities totally take place at the SAN level, i.e. we have $\mathcal{F}_1$-modelling. The output is given in a form that can be understood at the level of model description, i.e. in the $\mathcal{R}_1$-domain. Thus, $\mathcal{V}_1$-evaluations can certainly be done with the tool. Since UltraSAN allows for the easy construction of hierarchical models, higher-level evaluations are also possible. The rewards are input at the SAN level and need to be expressed as a general C function over the markings and/or the activity completions. Consequently, the rewards are not derived from a performance model (see the comments made with Metasan). Thus, $\mathcal{V}_i$-evaluations ($i \geq 1$) are possible with UltraSAN.

Evaluation. UltraSAN provides a very general framework for building performability models. The tool supports a wide variety of numerical and simulative evaluation techniques. Since performance aspects as well as dependability aspects are dealt with, the potential of the tool is very large. The multiple run option allows for an easy execution of parametric studies.

4.11. Surf-2

Surf-2 has been developed at LAAS-CNRS as a high-level tool for dependability and performability evaluation [6] as a successor to Surf [18]. It has been implemented for SUN 4 systems running UNIX, using the languages C and ADA.

Model class and evaluation techniques. Models can either be Markov reward models or GSPNs. The latter models are automatically transformed to the former before the analysis is started. Steady-state measures, transient measures and MRTA-measures can be derived by the tool. Which techniques are implemented is unclear from [6].

Model input and output. Both the Markov reward models and the GSPNs are input graphically. Output can be represented using a so-called result formatter which allows for the graphical representation of the results. As with UltraSAN (version 2), in the model definitions one can refer to global variables. By specifying the range of these variables, one can easily do series of analyses which, in combination with the result formatter, allow for the easy derivation of graphs.

GMTF fitting. Surf-2 allows for $\mathcal{V}_0$-evaluations when the Markov reward model input is chosen and for $\mathcal{V}_1$-evaluations when the GSPN input option is chosen. Rewards are specified as a general function over the markings, i.e. not model based. In total, $\mathcal{V}_1$ evaluations are possible.

Evaluation. Surf-2 is a powerful tool for the execution of series of dependability and performability analyses. Based on the powerful GSPN concept every required model can be made. Graphical input and output eases the modelling task. A limited set of measures can be derived.

4.12. DyQNtool$^+$

The tool DyQNtool$^+$ has been designed by Haverkort [39] at the University of Twente, as a successor to DyQNtool [36]. DyQNtool$^+$ has been implemented on SUN 4 systems (UNIX), using the C programming language. For its operation, it makes use of the packages SPNP and Sharpe.

Model class and evaluation techniques. DyQNtool$^+$ is based on Markov reward models. As with DyQNtool, the models are specified along the lines of the dynamic queueing network concept [34,35]. Basically, these are queueing networks that have been specified up to some parameters which are varying in time. The performance aspects of the model are described by this parametrized queueing network. The actual parameter values are a function Φ of the marking of the SPN that describes the dependability aspects. The function Φ can be fairly general, allowing for case recognition, etc. The generation of the Markov chain from the SPN description, the application of the function Φ and the subsequent substitution in and solution of the queueing network models proceed completely automatically.

The class of Markov chains that can be used is the same as the class that can be used by SPNP since the SPN-part of the dynamic queueing network models is handled by that package. For the rewards, all kinds of performance measures can be used, as long as they can be derived from "Sharpe queueing networks" (or are a function of such values). The function Φ, which maps the markings of the stochastic Petri net to the parameters of the queueing network model, can be any C function.

The Markov chain derived by SPNP and the rewards derived via multiple performance analyses performed by Sharpe are combined to derive steady-state, transient and cumulative performability measures. For this purpose, the steady-state and transient probabilities and the cumulative state residence times are derived by SPNP.

Model input and output. The input to DyQNtool$^+$ are a number of C files. First, there is the CSPL description of the dependability model, directly usable by SPNP. Secondly, there is the description of the parametrized queueing network as a set of special C function calls. When this description is executed, it generates code for Sharpe. Thirdly, there is the C file describing the mapping from the SPN markings to the queueing network parameters. Finally, also by some special C function calls, the performance measures to be used as rewards are specified.

Given the above files, DyQNtool$^+$ generates the underlying Markov chain and the necessary Sharpe performance models. It then solves these performance models using Sharpe and combines them with the state probabilities (or residence times) derived by SPNP. It finally presents the output in terms of tables.

GMTF fitting. With DyQNtool$^+$ the performability models are completely specified at a level higher than the Markov reward model level, i.e. as with Penpet and Numas, also the rewards are derived from a model. The tool therefore allows for $\mathcal{F}_1$-modelling. The final results are presented in a way that can be understood without knowledge of the underlying Markov reward model. Therefore, DyQNtool$^+$ is a tool that allows for $\mathcal{V}_1$-evaluations.

Evaluation. DyQNtool$^+$ is based on the concept of dynamic queueing networks. As such, it is based on a framework in which both the dependability and the performance aspects as well as their interdependence are formally specified. The limitation to "Sharpe queueing networks" is of a practical nature and not fundamental; moreover, it might even be considered to allow for the inclusion of other modelling options of Sharpe for the derivation of rewards. The derivation of large models becomes relatively easy with DyQNtool$^+$; the "hand work" needed to upgrade a Markovian dependability model to a Markov reward model suited for performability analysis is done totally automatically.

5. Summary and outlook

Due to the advent of fault-tolerant and distributed computer and communication systems, the interest in the combined performance and dependability modelling has steadily increased over the last decade. Many mathematical techniques have been developed to derive various measures from Markov reward models which form the basis of almost all performability models. We have briefly discussed these measures and models in Section 2 and provided many references to more detailed literature.

For evaluation techniques to be used, software tools are needed. We have introduced a general modelling tool framework in Section 3 that can guide the design of and the thinking about tools. From this framework, some general rules for the structuring of performability software tools can be derived:

(1) a modelling tool should optimally be adaptable to higher levels of abstraction, i.e. it should be possible to tailor the input formalism (and the output formalism) to specific application areas;
(2) lower level representations of models should be as transparent as possible;
(3) a modelling tool should have an input and an output formalism that are balanced, i.e. the output should be understandable when no more than the highest level model input is known.

We then, in Section 4, evaluated 12 software tools that can be used for performability modelling and analysis. From this survey, which was presented in historical order, we can conclude that:

(1) over the years there has been a shift from lower-level model formalisms, i.e. normally the Markov reward models, to higher-level formalisms;
(2) this shift is still more apparent in the dependability model part than in the performance model part and the output part;
(3) only the tools Numas, Penpet and DyQNtool$^+$ automatically derive and insert the rewards from a *model*; with Sharpe a similar functionality can be obtained, albeit in a more cumbersome way;
(4) in some tools, the rewards can be specified "at the net level", i.e. SPNP, UltraSAN and Surf-2, however, only as a general C function and not via a model;
(5) tools based on net models (Metasan, Penpet, UltraSAN, Surf-2) have proven to be the very flexible and generally applicable for performability modelling;
(6) the object-oriented approach towards systems modelling seems to be very general and easy extendible, since it is open ended towards various modelling approaches.

There is still a lot of work to do in the field of performability modelling and analysis. Recently, Meyer [58] indicated quite a large number of future work areas in the field of performability. Without repeating them all here, we just emphasize some important research areas that are especially related to tools for performability. Without trying to be exhaustive, we think that research is needed in the following directions:

(1) more tool support for hierarchical performability modelling formalisms that allows users to tailor their tool towards their specific application domains;
(2) tool support for the high-level specification of the rewards to be used in and the measures to be derived from these models;
(3) techniques to exploit the similarity in the series of performance models to be solved for obtaining the rewards;
(4) tool support for automatically handling large models, e.g. by using (approximate) truncation heuristics [37,89], exact lumping techniques [20], state space aggregation techniques [63], folding techniques [43] or fixed-point iteration techniques [14,17].

References

[1] H.H. Ammar, S.M.R. Islam and S. Deng, Performability analysis of parallel and distributed algorithms, *Proc. PNPM'89*, IEEE Computer Soc. Press, 1989, pp. 240–248.

[2] M.D. Beaudry, Performance related reliability measures for computing systems, *IEEE Trans. Comput.* **27**(6) (1978) 540–547.

[3] J. Bechta Dugan, K.S. Trivedi, R. Geist and V.F. Nicola, Extended stochastic Petri nets: Application and analysis, *Proc. Performance'84*, North-Holland, 1985.

[4] H. Beilner, Workload characterization and performance modelling tools, Paper presented at the *Int. Workshop on Workload Characterization of Computer Systems*, Pavia, Italy, October 23–25, 1985.

[5] H. Beilner, J. Mäter and N. Weissenberg, Towards a performance modelling environment: News on Hit, in: D. Potier and R. Puigjaner (Eds.), *Modelling Techniques and Tools for Computer Performance Evaluation*, Plenum Press, New York (1989) 57–75.

[6] C. Béounes, M. Aguéra, J. Arlat, S. Bachmann, C. Bourdeau, J.-E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell and P. Spiesser, Surf-2: A program for dependability evaluation of complex hardware and software systems, *Proc. FTCS 23*, IEEE Computer Soc. Press., 1993, pp. 668–673.

[7] S. Berson, E. de Souza e Silva and R.R. Muntz, An object oriented methodology for the specification of Markov models, UCLA Technical Report CSD-870030, 1987.

[8] M. Bouissou, The Figaro dependability evaluation workbench in use: Case studies for fault-tolerant computer systems, *Proc. FTCS 23*, IEEE Computer Soc. Press, 1993, pp. 680–685.

[9] J.A. Carrasco, Modelación y Evaluación de la Tolerancia a Fallos de Sistemas Distribuidos con Capacidad de Reconfiguracion, Ph.D. Thesis, University of Catalunya, Spain, 1986.

[10] J.A. Carrasco and J. Figueras, Metfac: Design and implementation of a software tool for modeling and evaluation of complex fault-tolerant computing systems, *Proc. FTCS 16*, IEEE Computer Soc. Press, 1986, pp. 424–429.

[11] *CCITT Blue Book, Fascicle III.1*, International Telecommunication Union, Geneva, 1989.

[12] G. Chiola, A graphical Petri net tool for performance analysis, in: S. Fdida and G. Pujolle (Eds.), *Modelling Techniques and Performance Evaluation*, North-Holland, Amsterdam (1987) 323–333.

[13] G. Chiola, C. Dutheillet, G. Franceschines and S. Haddad, Stochastic well-formed colored nets and symmetric modelling applications, *IEEE Trans. Software Engrg.* **42**(11) (1993) 1343–1360.

[14] H. Choi and K.S. Trivedi, Approximate performance models of polling systems using stochastic Petri nets, *Proc. INFOCOM'92*, IEEE Computer Soc. Press, 1992, pp. 2306–2314.

[15] G. Ciardo, J. Muppala and K.S. Trivedi, SPNP: Stochastic Petri net package, *Proc. PNPM'89*, IEEE Computer Soc. Press, 1989, pp. 142–151.

[16] G. Ciardo, J.K. Muppala and K.S. Trivedi, On the solution of GSPN reward models, *Performance Evaluation* **12**(4) (1991) 237–254.

[17] G. Ciardo and K.S. Trivedi, A decomposition approach for stochastic reward net models, *Performance Evaluation* **18** (1993) 37–59.

[18] A. Costes, J.-E. Doucet, C. Landrault and J.-C. Laprie, Surf: A program for dependability evaluation of complex fault-tolerant computing systems, *Proc. FTCS 11*, IEEE Computer Soc. Press, 1981, pp. 72–78.

[19] P.-J. Courtois, *Decomposability, Queueing and Computer Science Applications*, Academic Press, New York (1977).

[20] J.A. Couvillion, R. Freire, R. Johnson, W.D. Obal II, A. Qureshi, M. Rai, W.H. Sanders and J.E. Tvedt, Performability modelling with UltraSAN, *IEEE Software* (September 1991) 69–80.

[21] N.M. van Dijk, B.R. Haverkort and I.G. Niemegeers, Guest editorial: Performability modelling of computer and communication systems, *Performance Evaluation* **14**(3,4) (1992) 135–138.

[22] L. Donatiello and B.R. Iyer, Analysis of a composite performance reliability measure for fault-tolerant systems, *J. ACM* **34**(1) (1987) 179–199.

[23] D.G. Furchtgott and J.F. Meyer, A performability solution method for degradable non-repairable systems, *IEEE Trans. Comput.* **33**(6) (1984) 550–554.

[24] D.G. Furchtgott, Performability models and solutions, Ph.D. Dissertation, The University of Michigan, 1984.

[25] A. Goyal, W.C. Carter, E. de Souza e Silva, S.S. Lavenberg and K.S. Trivedi, The system availability estimator, *Proc. FTCS 16*, IEEE Computer Soc. Press, 1986, pp. 84–89.

[26] A. Goyal and A.N. Tantawi, Evaluation of performability for degradable computer systems, *IEEE Trans. Comput.* **36**(6) (1987) 738–744.
[27] A. Goyal, S.S. Lavenberg and K.S. Trivedi, Probabilistic modelling of computer system availability, *Ann. Oper. Res.* **8** (1987) 285–306.
[28] A. Goyal and S.S. Lavenberg, Modelling and analysis of computer system availability, *IBM J. Res. Develop.* **31**(6) (1987) 651–664.
[29] A. Goyal, System availability estimator – User's Manual, Version 2.0, Internal Report, IBM Yorktown Heights, NY, USA, 1987.
[30] A. Goyal, P. Shahabuddin, P. Heidelberger, V.F. Nicola and P.W. Glynn, A unified framework for simulating Markovian models of highly dependable systems, *IEEE Trans. Comput.* **41**(1) (1992) 36–51.
[31] W. Grassmann, Means and variances of time averages in Markovian environments, *European J. Oper. Res.* **31**(1) (1987) 132–139.
[32] D. Gross and D.R. Miller, The randomization technique as a modelling tool and solution procedure for transient Markov processes, *Oper. Res.* **32**(2) (1984) 343–361.
[33] J.F. Meyer, Performability evaluation: where it is and what lies ahead, *Proc. IEEE Int. Computer Performance and Dependability Symp.*, 1995, pp. 334–343.
[34] B.R. Haverkort and I.G. Niemegeers, Using dynamic queueing networks for performability modelling, *Proc. European Simulation Multiconference 1990*, Society for Computer Simulation, 1990, pp. 184–191.
[35] B.R. Haverkort, Performability modelling tools, evaluation techniques, and applications, Ph.D. Thesis, University of Twente, 1990.
[36] B.R. Haverkort, I.G. Niemegeers and P. Veldhuyzen van Zanten, DyQNtool—A performability modelling tool based on the dynamic queueing network concept, in: G. Balbo and G. Serazzi (Eds.), *Computer Performance Evaluation: Modelling Techniques and Tools*, North-Holland, Amsterdam (1992) 181–195.
[37] B.R. Haverkort, Approximate performability and dependability modelling using generalized stochastic Petri nets, *Performance Evaluation* **18**(1) (1993) 61–78.
[38] B.R. Haverkort and K.S. Trivedi, Specification and generation of Markov reward models, in: *Discrete-Event Dynamic Systems: Theory and Applications* **3** (1993) 219–247.
[39] B.R. Haverkort, Performability modelling using DyQNtool$^+$, *Int. J. Reliability, Quality Safety Engrg.* **2**(4) (1995).
[40] B.R. Haverkort, A.P.A. van Moorsel and D.-J. Speelman, Xmgm: A performance analysis tool based on matrix geometric methods, *Proc. MASCOTS'94*, IEEE Computer Soc. Press, 1994, pp. 152–157.
[41] P. Heidelberger and A. Goyal, Sensitivity analysis of continuous time Markov chains using uniformization, in: G. Iazeolla, P.-J. Courtois and O.J. Boxma (Eds.),*Computer Performance and Reliability*, North-Holland, Amsterdam (1988) 93–104.
[42] R.A. Howard, *Dynamic Probabilistic Systems, Vol. 2: Semi-Markov and Decision Processes*, Wiley, New York (1971).
[43] O.C. Ibe, H. Choi and K.S. Trivedi, Performance evaluation of Client-Server systems, *IEEE Trans. Parallel Distrib. Systems* **4**(11) (1993) 1217–1229.
[44] A.M. Johnson Jr. and M. Malek, Survey of software tools for evaluating reliability, availability, and serviceability, *ACM Comput. Surveys* **20**(4) (1988) 227–269.
[45] G. Klas, Hierarchical evaluation of generalized stochastic Petri nets, Ph.D. Thesis, Technical University, München, 1993.
[46] U. Krieger, B. Müller-Clostermann and M. Sczittnick, Modelling and analysis of communication systems based on computational methods for Markov chains, *IEEE J. Selected Areas Comm.* **8**(9) (1990) 1630–1648.
[47] J.C. Laprie, Dependable computing and fault-tolerance: Concepts and terminology, *Proc. FTCS 15*, IEEE Computer Soc. Press, 1985, pp. 2–7.
[48] R. Lepold, Tompsin: Benutzerhandbuch, Internal Report Siemens AG, 1991.
[49] R. Lepold, Penpet: A performability modelling evaluation tool based on stochastic Petri nets, *Proc. Joint Int. Meeting of TIMS XXX and SOBRAPO XXIII*, Rio de Janeiro, 1991.
[50] R. Lepold, Performability evaluation of a fault-tolerant computer systems using stochastic Petri nets, *Proc. 5th Int. Conf. on Fault-Tolerant Computing Systems*, Nürnberg, 1991.
[51] R. Lepold, Performability evaluation of degradable computer systems based on stochastic Petri nets, Ph.D. Thesis, Université de Haute Elsace, France, 1992.
[52] C. Lindemann and R. German, DSPNexpress: A software package for efficiently solving deterministic and stochastic Petri nets, in: R. Pooley and J. Hillston (Eds.), *Performance Tools 1992*, Edinburgh University Press, Edinburgh (1993).

[53] R.A. Marie, A.L. Reibman and K.S. Trivedi, Transient analysis of acyclic Markov chains, *Performance Evaluation* **7** (1987) 175–194.
[54] H. de Meer, Transiente Leistungsbewertung und Optimierung Rekonfigurierbarer Fehlertoleranter Rechensysteme, Ph.D. Thesis, Friedrich-Alexander Universität Erlangen-Nürnberg, 1992.
[55] J.F. Meyer, On evaluating the performability of degradable computer systems, *IEEE Trans. Comput.* **29**(8) (1980) 720–731.
[56] J.F. Meyer, Closed-form solutions of performability, *IEEE Trans. Comput.* **31**(7) (1982) 648–657.
[57] J.F. Meyer, A. Movaghar and W.H. Sanders, Stochastic activity networks: Structure, behaviour, and application, *Proc. PNPM'85*, IEEE Computer Soc. Press, 1985, pp. 106–115.
[58] J.F. Meyer, Performability: A retrospective and some pointers to the future, *Performance Evaluation* **14**(3,4) (1992) 139–156.
[59] A.P.A. van Moorsel, Performability evaluation concepts and techniques, Ph.D. Thesis, University of Twente, 1993.
[60] B. Müller-Clostermann, Numas – A tool for the numerical analysis of computer systems, in: D. Potier (Ed.), *Proc. Int. Conf. on Modelling Techniques and Tools for Performance Analysis*, North-Holland, 1995, pp. 141–154.
[61] M. Mulazzani and K.S. Trivedi, Dependability prediction: Comparison of tools and techniques, *Proc. IFAC Safecomp*, 1986, pp. 171–178.
[62] M. Mulazzani, An open layered architecture for dependability analysis and its applications, *Proc. FTCS 18*, IEEE Computer Soc. Press, 1988.
[63] R.R. Muntz, E. de Souza e Silva and A. Goyal, Bounding availability of repairable computer systems, *IEEE Trans. Comput.* **38**(12) (1989) 1714–1723.
[64] J.K. Muppala and K.S. Trivedi, Numerical transient solution of finite Markovian queueing systems, in: U.N. Bhat and I.V. Basawa (Eds.), *Queueing and Related Models*, Oxford University Press, Oxford (1992) 262–284.
[65] W.D. Obal II and W.H. Sanders, Importance sampling in UltraSAN, Simulation **62**(2) (1994) 98–111.
[66] T.W. Page. Jr., S.E. Berson, W.C. Cheng and R.R. Muntz, An object-oriented modelling environment, *ACM Sigplan Notices* **24**(10) (1989) 287–296.
[67] K.R. Pattipati, Y. Li and H.A.P. Blom, A unified framework for the performability evaluation of fault-tolerant computer systems, *IEEE Trans. Comput.* **42**(3) (1993) 312–326.
[68] M.A. Qureshi and W.H. Sanders, Reward model solution methods with impulse and rate rewards: An algorithm and numerical results, *Performance Evaluation* **20**(4) (1994) 413–436.
[69] A.V. Ramesh and K.S. Trivedi, Semi-numerical transient analysis of Markov models, Technical Report DUKE-CCSR-92-014, Department of Electrical Engineering, Duke University, 1992.
[70] A.L. Reibman and K.S. Trivedi, Numerical transient analysis of Markov models, *Comput. Oper. Res.* **15**(1) (1988) 19–36.
[71] A.L. Reibman, R. Smith and K.S. Trivedi, Markov and Markov reward models transient analysis: An overview of numerical approaches, *European J. Oper. Res.* **40** (1989) 257–267.
[72] A.L. Reibman and K.S. Trivedi, Transient analysis of cumulative measures of Markov model behavior, *Stochastic Models* **5**(4) (1989) 683–710.
[73] R.A. Sahner, A Hybrid, Combinatorial-Markov method for solving performance and reliability models, Ph.D. Dissertation, Duke University, 1986.
[74] R.A. Sahner and K.S. Trivedi, A hierarchical, Combinatorial-Markov method of solving complex reliability models, *Proc. Fall Joint Computer Conference*, IEEE Computer Soc. Press, 1986, pp. 817–825.
[75] R.A. Sahner and K.S. Trivedi, Reliability modelling using sharpe, *IEEE Trans. Reliability* **36**(2) (1987) 186–193.
[76] R.A. Sahner and K.S. Trivedi, Performance and reliability analysis using directed acyclic graphs, *IEEE Trans. Software Engrg.* **13**(10) (1987) 1105–1114.
[77] R.A. Sahner and K.S. Trivedi, A software tool for learning about stochastic models, *IEEE Trans. Ed.*, **36**(1) (1993) 56–61.
[78] W.H. Sanders and J.F. Meyer, Metasan: A performability evaluation tool based on stochastic activity networks, ITI Technical Report 85-22.1, 1986 (presented at the *IEEE-ACM Fall Joint Computer Conference*, Dallas, TX, November 1986).
[79] W.H. Sanders and J.F. Meyer, Performability evaluation of distributed systems using stochastic activity networks, *Proc. PNPM'87*, IEEE Computer Soc. Press, 1987, pp. 111–120.
[80] W.H. Sanders and J.F. Meyer, Performance variable driven construction methods for stochastic activity networks, in: G. Iazeolla, P.-J. Courtois and O.J. Boxma (Eds.), *Computer Performance and Reliability*, North-Holland, Amsterdam (1988) 383–398.

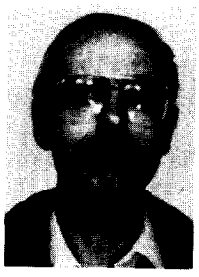
[81] W.H. Sanders, Construction and solution of performability models based on stochastic activity networks, Ph.D. Dissertation, University of Michigan, USA, 1988.
[82] W.H. Sanders and J.F. Meyer, Reduced base model construction for stochastic activity networks, *IEEE J. Selected Areas Comm.* **9**(1) (1991) 25–36.
[83] W.H. Sanders, W.D. Obal II, M.A. Qureshi and F.K. Widjanarko, UltraSAN version 2: Architecture, features and implementation, PMRL Technical Report 93-17, University of Arizona, 1993 (submitted for publication).
[84] M. Sczittnick and B. Müller-Clostermann, Macom—A tool for the Markovian analysis of communication systems, *Proc. 4th Int. Conf. on Data Communication Systems and Their Performance*, 1990, pp. 456–470.
[85] R.M. Smith, K.S. Trivedi and A.V. Ramesh, Performability analysis: Measures, an algorithm and a case study, *IEEE Trans. Comput.* **37**(4) (1988) 406–417.
[86] E. de Souza e Silva and H.R. Gail, Calculating cumulative operational time distributions of repairable computer systems, *IEEE Trans. Comput.* **35**(4) (1986) 322–332.
[87] E. de Souza e Silva and H.R. Gail, Calculating availability and performability measures of repairable computer systems using randomization, *J. ACM* **36**(1) (1989) 171–193.
[88] E. de Souza e Silva and H.R. Gail, Performability analysis of computer systems: From model specification to solution, *Performance Evaluation* **14**(3,4) (1992) 157–196.
[89] E. de Souza e Silva and P.M. Ochoa, State space exploration in Markov models, *ACM Performance Evaluation Rev.* **20**(1) (1992) 152–166.
[90] W.J. Stewart and A. Goyal, Matrix methods in large dependability models, IBM Research Report RC 11485, 1985.
[91] H. Tardif, K.S. Trivedi and A.V. Ramesh, Closed-form transient analysis of Markov chains, Technical Report CS-1988, Dept. of Computer Science, Duke University, Durham, NC, 27706, June 1988.
[92] K.S. Trivedi, J.K. Muppala, S.P. Woolet and B.R. Haverkort, Composite performance and dependability analysis, *Performance Evaluation* **14**(3,4) (1992) 197–215.

Boudewijn R. Haverkort received the M.Sc. and the Ph.D. degree in Computer Science, both from the University of Twente, Enschede, The Netherlands, in 1986 and 1991, respectively.

Since October 1995, he is a professor at the Rheinische Westfälische Technische Hochschule Aachen, Germany, responsible for the area of distributed systems. From January 1990 through September 1995 he was assistant professor in the Tele-Informatics and Open Systems group of the department of Computer Science at the University of Twente. From January through April 1993 he was a visiting Professor at the Teletraffic Research Centre of the University of Adelaide, Australia.

His current research interests include the design and (performability) evaluation of fault-tolerant and distributed computer and communication systems.

In 1992, he served as a guest-editor for a special issue of *Performance Evaluation* devoted to performability modelling of computer and communication systems. Dr. Haverkort, who is a member of the IEEE Computer and Communication Societies and the ACM, received a Koninklijke/Shell research prize early 1991.



Ignas G. Niemegeers received the Electrical Engineering degree from the University of Gent, Belgium, in 1970, and the M.Sc.E. degree and the Ph.D. degree in Computer Engineering, both from Purdue University, West-Lafayette, Indiana, USA, in 1972 and 1978, respectively. From 1978 to 1981 he was a designer of packet switching networks at Bell Telephone Manufacturing Company, Antwerp, Belgium.

Since 1981 he has been a Professor at the University of Twente, Enschede, The Netherlands, in both the Electrical Engineering and the Computer Science department. He presently is Scientific Director of the Centre for Telematics and Information Technology, a multi-disciplinary research institute at the University of Twente.

His areas of interest include communication systems and performance analysis. He is in active research on integrated networking, high-speed networking, B-ISDN, optical switching, performance analysis and performability.