appears in: Proc. of the 2nd Workshop on Process Algebras and Performance Modelling,
Regensberg, 21.–22. July, 1994

Stochastic Process Algebras Applied to Failure Modelling

Ulrich Herzog and Vassilis Mertsiotakis

University of Erlangen-Nürnberg, IMMD VII, Martensstr. 3, 91058 Erlangen, Germany
{herzog,vsmertsi}@informatik.uni-erlangen.de

Abstract

In the past we concentrated on the conceptual design and theoretical foundation of stochastic process algebras and applied this promising method mainly to the performance evaluation of multiprocessor and distributed systems. Nevertheless, we demonstrated at the READPAC–Workshop in principle and by an example how to apply these ideas to the description of reliability aspects. Our current version of TIPP allows us to extend these results. In particular our prototype evaluation tool now provides efficient evaluation of both performance and dependability measures for quite complex systems.

1 Introduction

Dependability analysis and performance evaluation of computer systems have been carried out separately for a long time. The idea of integrating reliability modelling and performance modelling, however, is not new. The examination of reliability with queueing systems for example was presented already in [40]. Various combined measures for performance and dependability were proposed in [2]. A general framework for performability analysis was presented in [28]. A more detailed overview of this area may be found in [39].

Another important direction in this field is the design of new modelling techniques for performability analysis as well as the integration of dependability aspects into existing paradigms and tools. An extension of Generalized Stochastic Petri Nets towards performability was introduced in [8]. Trivedi and Haverkort present in [19] an overview of modelling techniques and tools that are suitable for performability analysis. Our aim is to motivate the use of stochastic process algebras for performability analysis.

Specification techniques capturing all main characteristics of system behaviour are more costly than classical methods where separate models are used to describe the functional behaviour, the performance characteristics, and the reliability aspects. More costly means that:

- all aspects of quality have to be considered and specified at the same time;

- the underlying theory is more demanding and the corresponding models are more complex.

This is true for all advanced techniques including Stochastic Automata, Stochastic Petri Nets and Stochastic Process Algebras. In order to compensate for these drawbacks, the techniques have to be used extensively. Then, however, their obvious advantages come to light:

- more accurate description of system behaviour,
- more comprehensive system analysis, and
- formal deduction for models and quality measures from specifications amenable to automation efforts, i.e.
- improved design productivity, i.e. faster and less error prone realization of hardware – and software products.

In the past we concentrated mainly on both the functioning and performance of systems. This paper shows how to apply Stochastic Process Algebras also to the modelling of reliability, availability, and performability aspects.

As parallel computers and communication systems become physically and logically more complex, their performance is often degradable, i.e. internal or external faults can reduce the quality of a delivered service significantly. Both hardware- and software architectures supporting graceful degradation, fault tolerance, and fault avoidance are therefore very important. This is true also for related modeling and evaluation techniques and an excellent overview on the state-of-the-art may be found in [18]. Typical questions are e.g.: "How long can the system be expected to work without interruption?; How much work can the system be expected to accomplish before a failure?; How it is possible to optimize reconfigurable fault tolerant systems [13], [10].

To answer these questions, the most commonly employed methods are analytical methods and simulation. Work on our process algebra TIPP concentrates on analytical methods for analysis purposes. An alternative approach, namely to apply simulation to the analysis of timed process algebras is presented in [34].

We analyze the performability of a TIPP process by deriving a Continuous Time Markov Chain (CTMC) from a TIPP process and applying well known analysis algorithms. Unfortunately, modelling dependability aspects leads very often to a so-called *stiff* Markov chain and analysis becomes very computationally expensive. When using simulation, a similar problem arises, known as the problem of *rare events*. Several methods are known from the literature to overcome this problem to some extent. One of the most promising approaches is to use special solution methods based on the aggregation–disaggregation algorithm of *Courtois* [9]. In order to apply such methods successfully, information about the model structure is needed. The use of high level modelling techniques allows us to exploit structural information provided with a model. Ammar and Islam examined this problem for stochastic Petri nets [1]. In this paper we will show how this approach in principle can be adopted for stochastic process algebras.

This paper is organized as follows. In the next section we will present briefly the Stochastic Process Algebra TIPP (Timed Processes and Performance Evaluation) developed by our group at the University of Erlangen. Section 3 shows how to model dependability aspects with TIPP. We will address some common problems that occur usually in performability analysis and sketch

possible solutions known from the literature and how to exploit the process algebra approach in order to tackle these problems. Finally, in section 4 an extensive case study is presented and section 5 concludes the paper.

2 The Stochastic Process Algebra TIPP

We briefly summarize the basic ideas of syntax, semantics and model evaluation of the current version of TIPP. More details may be found in our tutorial paper [15] and recent research reports. The reader is expected to be familiar with the basic concepts and standard notation of process algebras, performance evaluation and reliability theory. Excellent introductions may be found in [23], [29], [25], and [38].

The process algebra TIPP – timed processes and performance/performability evaluation – is an extension of the classical abstract languages CSP [23] and CCS [29] including random time variables. The basic elements for descriptions are activities and combination operators. Actions describe relevant activities of the system; they are considered to be atomic. Descriptions can be composed in order to yield descriptions of more complex systems. Formally, the ways descriptions can be built are defined by a grammar.

We assume a fixed set of action names $Act := Com \cup \{\tau\}$, where we use τ as a distinguished symbol for internal, invisible actions and Com as the set of regular, visible activities (the communication actions). An action a can either be passive $(a, \mathbb{1})$ or exponentially distributed.

Definition 2.1 *The set $\mathcal{L}$ of terms of the language is given by the following grammar [21]:*

$$P \;::=\; 0 \;\mid\; (a,\lambda).P \;\mid\; P + P \;\mid\; P\|_S P \;\mid\; P\backslash a \;\mid\; rec\, X : P \;\mid\; X \;,$$

where $a \in$ Act, $\lambda \in \mathbb{R}^+$; $S \subseteq$ Act$\backslash\{\tau\}$, $X \in$ Var, and Var is a set of process variables.

The intuitive meaning of these basic elements and operators is as follows: 0 denotes the halting process, $(a, \lambda).P$ prefixing; next follow the choice operator, the parallel operator including a set S of synchronizing actions. We also include the hiding operator and the recursion operator allowing the description of infinite behaviour.

For presenting a semantics for our language we adopted the Structural Operational Semantics (SOS) style introduced by Plotkin: With each Process $P \in \mathcal{L}$ the semantics associates a transition system

$$(\ \mathcal{L}\ ,\ P\ ,\ \longrightarrow\ )$$

where

$$\longrightarrow \in\ \mathcal{L} \times (Act \times \mathbb{R}^+ \times \{\epsilon, l, r\}) \times\ \mathcal{L}$$

is the least relation that satisfies the rules of Fig. 1.

Again, more detailed explanations, motivations and justifications for TIPP can be found in [15, 21].

$$\langle . \rangle \quad \frac{}{(a,\lambda).P \xrightarrow{a,\lambda,\varepsilon} P} \qquad \langle \|_l \rangle \quad \frac{P \xrightarrow{a,\lambda,w} P'}{P \|_S Q \xrightarrow{a,\lambda,\|_l.w} P' \|_S Q} \quad (a \notin S)$$

$$\langle +_l \rangle \quad \frac{P \xrightarrow{a,\lambda,w} P'}{P + Q \xrightarrow{a,\lambda,+_l.w} P'} \qquad \langle \|_r \rangle \quad \frac{Q \xrightarrow{a,\lambda,w} Q'}{P \|_S Q \xrightarrow{a,\lambda,\|_r.w} P \|_S Q'} \quad (a \notin S)$$

$$\langle +_r \rangle \quad \frac{Q \xrightarrow{a,\lambda,w} Q'}{P + Q \xrightarrow{a,\lambda,+_r.w} Q'} \qquad \langle \| \rangle \quad \frac{P \xrightarrow{a,\lambda,v} P' \quad Q \xrightarrow{a,\mu,w} Q'}{P \|_S Q \xrightarrow{a,\lambda\mu,(v,w)} P' \|_S Q'} \quad (a \in S)$$

$$\langle \backslash_{yes} \rangle \quad \frac{P \xrightarrow{a,\lambda,w} P'}{P\backslash a \xrightarrow{\tau,\lambda,w} P'\backslash a} \qquad \langle \backslash_{no} \rangle \quad \frac{P \xrightarrow{b,\lambda,w} P'}{P\backslash a \xrightarrow{b,\lambda,w} P'\backslash a} \quad (a \neq b)$$

$$\langle rec \rangle \quad \frac{P\{(recX : P)/X\} \xrightarrow{a,\lambda,w} P'}{recX \ : \ P \xrightarrow{a,\lambda,w} P'}$$

Figure 1: Operational Semantics of TIPP [21]

3 Modelling Unreliable Behaviour

3.1 Examples for Unreliable Systems

The Stochastic Process Algebras allow the modelling of both the functional and temporal aspects of a system. Unreliable behaviour means that a system or some components of it may function for some time and then may fail. The choice operator allows the expression of such a behaviour and two examples may illustrate this:

- An arrival process which may be interrupted after some (exponentially distributed time with rate ϵ) is given by

  $$Arriv := (in, \lambda).Arriv \ + \ (fail, \epsilon).0$$

  Similarly, a processor working in a batch processing mode may fail after some time and then stop working

  $$Proc := (work, \nu).Proc \ + \ (fail, \epsilon).0$$

- A processor waiting for jobs to be processed may also fail and then goes to repair

  $$Proc := (in, \mathbb{1}).Busy \ + \ (fail, \epsilon).Repair$$

  Of course, the busy processor may fail also while processing a job

  $$Busy := (work, \nu).Proc \ + \ (fail, \epsilon).Repair$$

  After failure error detection routines may be started and the faulty component repaired

  $$Repair := (det, \delta_1).(rep, \delta_2).Proc$$

Of course, structured design of complex systems is supported by the parallel composition operator which may be used efficiently also for describing complex fault tolerant systems. Again, the basic idea is shown by simple examples.

- Consider some workloads W_A and W_B mapped onto two unreliable processors $Proc_A$ and $Proc_B$ respectively. Then, the overall system behaviour is described by the parallel compostion

$$SM := (W_A \parallel_A Proc_A) \parallel_\emptyset (W_B \parallel_B Proc_B)$$

  where A and B are the synchronizing actions mapping the workload onto the two processors. Note that the processors may stop processing or resume work after repair, dependent on the underlying processor models.

- Of course, remapping of workload in case of processor failure can also be modelled and a simple example is shown next

$$\begin{aligned}
SM &:= SM_A \parallel_{\{fail_A, fail_B\}} SM_B \\
SM_A &:= (work_A, \mu).SM_A + (fail_A, \epsilon).0 + (fail_B, \mathbb{1}).SM_{AB} \\
SM_B &:= (work_B, \mu).SM_B + (fail_B, \epsilon).0 + (fail_A, \mathbb{1}).SM_{AB} \\
SM_{AB} &:= (work_{AB}, \mu).SM_{AB} + (fail_{AB}, \epsilon).0
\end{aligned}$$

- Consider a multiprocessor system with n processors and unlimited workload [17]. Assume that failures occur with rate $i\gamma$, if i processors are up and j jobs are processed with rate μ. Processor failures are covered with probability c and not covered with probability $\overline{c} = 1 - c$ [17]. After a covered failure the system is reconfigured quickly (with rate δ) and comes up in a degraded mode. A longer reboot action is required (rate β) if the failure is uncovered. The repair of a processor may be completed with rate τ. All time intervals are assumed to be exponentially distributed with the corresponding rates. Furthermore we assume that during reconfiguration of a reboot no other events can occur since these times are extremely small compared to times related to failures and repair. The behaviour of the processor may be specified as follows:

$$\begin{aligned}
MP_n &:= (proc, \mu).MP_n \\
&\quad + (covered_failure, i\gamma c).Reconf_n \\
&\quad + (uncovered_failure, i\gamma\overline{c}).Reboot_n \\
MP_i &:= (proc, \mu).MP_i \\
&\quad + (covered_failure, i\gamma c).Reconf_i \\
&\quad + (uncovered_failure, i\gamma\overline{c}).Reboot_i \\
&\quad + (repair, \mathbb{1}).MP_{i+1} \qquad (0 < i < n) \\
MP_0 &:= (repair, \mathbb{1}).MP_1
\end{aligned}$$

  If there is a single repair facility shared by all processors they may be repaired only one by one. In this most simple case the repair facility may be described by

$$Repair := (repair, \tau).Repair$$

Assuming that the multiprocessor starts in a undegraded operation mode, the overall system behaviour is described in the parallel composition of the multiprocessor and the repair facility

$$System := MP_n \parallel_{\emptyset} Repair$$

More sophisticated models are possible introducing special scheduling processes [14], introducing priorities [20] or making intensive use of immediate transitions [10].

3.2 Reliability and Performability Measures

Applying the operational semantics rules the underlying labelled transition system can be generated for each system description. This transition system contains all characteristic information and may be evaluated with respect to the functional behaviour, performance as well as reliability measures. We concentrate on the analysis of reliability and performability aspects and related measures. Hiding the action labels and removing immediate actions the transition system can systematically be reduced to a Markov chain model which can be described and analyzed via some standard notion and standard measures. Our compact description follows strictly the lines of [13]:

Let $\{X(t); t \geq 0\}$ be a homogeneous continuous time Markov chain and let $S = \{a_i; i = 1, \ldots, n\}$ be the finite state space associated with the model. It is also standard to assume $k + 1$ rewards $\rho_1 > \ldots > \rho_{k+1}$ which may be associated with states or transitions.

Let S_0 be the set of states that represent an operational system and S_F be the remaining set of states that represent a failed system

- Point availability

  Point availability $A(t)$ is defined as the probability that the system is operational at time t. Defining an indicator random variable

  $$I(t) = \begin{cases} 1 & \text{if} \quad X(t) \in S_0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

  the point availability is given by

  $$A(t) = P[I(t) = 1] = E[I(t)] \tag{2}$$

- Cumulative operational time

  The cumulative operational time $O(t)$ indicates the total amount of operational time during $(0, t)$:

  $$O(t) = \int_0^t I(s)ds \tag{3}$$

- Interval availability

  The interval availability represents the fraction of time a system is operational during $(0, t)$:

  $$A_I(t) = \frac{O(t)}{t} \tag{4}$$

- Reliability is defined as the probability that the system is operational during the entire observation period:

$$R(t) = 1 - \lim_{s \to t} P[O(t) \leq s] \tag{5}$$

- Lifetime $L(t)$ is equal to the time of the first system failure, if such occurs before t and equal to t otherwise.

$$L(t) = \int_0^t 1 - I(s)ds \tag{6}$$

- Mean time to failure MTTF is the limited expected lifetime

$$MTTF = \lim_{t \to \infty} E[L(t)] \tag{7}$$

To value different aspects of system behaviour, one often introduces rewards associated with the states and/or transitions of the CTMC. Then the model is called a reward model or if we include immediate transitions an extended reward model [10]. Some important measures are shown next:

- Point performability

  Let r_i be the reward associated with state a_i. The instantaneous reward at time t is then

$$Y(t) = \begin{cases} r_i & \text{if} \quad X(t) = a_i \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

  The point performability $M(t)$ is defined as its expected value

$$M(t) = E[Y(t)] = \sum_{i=1}^{n} r_i P[X(t) = a_i] \tag{9}$$

  Various classical measures can be derived from this point performability measure. For example, the point throughput of an action a can be obtained by:

$$r_i = \sum_j \lambda_j \quad \text{where} \quad (a, \lambda_j) \quad \text{is enabled in state } i \tag{10}$$

- Cumulative reward

  The cumulative reward during $(0, t)$ is defined by

$$Z(t) = \int_0^t Y(s)ds \tag{11}$$

More measures have been defined and may be found in the references mentioned above. Analyzing the models and determining characteristic quality measures several technical problems may occur such as state-space explosion and stiffness. Again, the referenced literature deals with these problems and shows solutions. We will briefly summarize the most important techniques and show how to apply them to stochastic process algebras in the rest of this section.

3.3 Solving the Underlying Markov Chain

Dependent on the measures to be computed, the basis for further analysis is either the steady state analysis of the underlying CTMC, or transient analysis, i.e. the examination of the system's time dependent behaviour. In the former, the following linear equation system, known as *Chapmann–Kolmogoroff equation system*, has to be solved in order to obtain the steady state probability distribution π:

$$\pi Q = 0 \quad \text{subject to} \quad \sum_{i=1}^{n} \pi_i = 1 \tag{12}$$

The matrix Q is the infinitesimal generator matrix of the CTMC underlying a LTS and can pretty straightforwardly be obtained by reducing parallel arcs and removing loops.

There are plenty of different solution methods for the above equations. Direct solution methods, like Gaussian elimination, LU–decomposition, or the Grassmann–method, are suitable for small systems with up to 2000 states and can by applied for every irreducible Markov chain. Iterative solution schemes, like the Power–method, Gauß–Seidel, or SOR, are much faster than direct methods, however, the convergence is not always assured [26].

More intelligent solution methods try to exploit information out of the model in order to reduce the solution effort. Usually, the knowledge of symmetries, regularities, or hierarchies in the model structure enable the use of very efficient algorithms, like the Matrix–Geometric solution method [31], the Spectral–Expansion method [7], Disaggregation methods [35] or Tensor–Algebra based methods [5, 32, 36] to name a few.

Buchholz showed recently in [6] how to adapt his method to a stochastic process algebra similar to TIPP. The adaption of Siegle's method to a subset of TIPP is shown in [33]. In Hillston's PhD–Thesis [22] and Hermanns' Masters–Thesis [20] it is shown how to aggregate equivalent states using notions of equivalence between processes for PEPA and TIPP respectively. Hillston and Buchholz have examined thoroughly the relation to lumpability of Markov chains.

If steady state analysis is not possible or if time dependent quantities are desired, like the *Point Availability* or *Point Performability*, transient analysis has to be carried out by solving the *Chapmann–Kolmogoroff* differential equation system:

$$\frac{d\pi(t)}{dt} = \pi(t)Q \quad \text{where} \quad \pi(0) = e_1 \tag{13}$$

Although the above differential equation system has a closed solution

$$\pi(t) = \pi(0)e^{Qt} = \sum_{k=0}^{\infty} \pi(0)\frac{(Qt)^k}{k!} \tag{14}$$

this way is often avoided because of its numerical instability [16]. Instead, the *randomization technique* is used very frequently, which is based on the transformation of Q to the stochastic matrix of an embedded discrete time Markov chain. Among others, this approach was adopted also by Lindemann [27], who presented a refined randomization technique based on a numerically stable algorithm for the computation of Poisson–probabilities [12].

Unfortunately, in addition to largeness another well known problem arises, especially in performability models, namely the problem of having to solve a stiff differential equation system. Typically,

such systems are caused by transition rates that differ in many orders of magnitude. The above mentioned refined randomization scheme is able to tackle this problem to some extent. Nevertheless, the solution effort remains still relatively high. The use of implicit integration methods is another possibility, but this method is not able either to reduce the solution effort. Therefore, Trivedi et. al. [3, 4] have proposed applying aggregation techniques based on the approximate decomposition method of Courtois [9] to transient analysis of stiff Markov chains. The characterization of a class of TIPP–Models that could be analyzed using such aggregation techniques is presented next.

3.3.1 Decomposable processes

Many algorithms for the computation of the steady–state solution of continuous time or discrete time Markov chains are based on the decomposition of the state space into several distinct partitions. By solving the resulting smaller Markov chains, an approximate solution of the whole model can be obtained using the aggregation technique of Courtois [9]. There are various iterative aggregation methods known which are able to estimate the correct solution very accurately by combining the aggregation technique with iterative solution methods, like SOR, Jacobi, Block–SOR, etc. [37, 26]. Recently, a new approach of this kind based on multigrid methods was presented in [24].

As the quality of the aggregated solution and the required computation effort for these methods depends strongly on the model structure, we present a characterization of TIPP–processes that could be analyzed efficiently using such methods:

Definition 3.1 *Consider $P \in \mathcal{L}$, a process with finite state space. P is called* LM–decomposable, *if*

$$P = L \parallel_S M \quad \wedge \quad (\forall a \in S)(L \xrightarrow{a,\cdot} L' \Rightarrow L = L') \tag{15}$$

which denotes that the process L can only change its state while it is not interacting with the process M. Further, we require that the process L has also a finite state space and that no free variables occur within L.

Note that the processes L and M might as well be parallel compositions of other processes. At the first look the above conditions seem to be very restrictive. However, there is a large variety of applications that fit this definition. As soon as the mapping of a workload to a machine model has to be modelled, where the workload has different arrival rates – in queueing theory commonly referred to as Markov Modulated Poisson Processes [30] – this may easily be modelled by a LM–decomposable process. Actually, the definition of *LM–decomposable* processes was motivated by this class of *workload-to-machine-mapping* models. A more general class of decomposable processes, where explicit modelling of workload is not required, will be presented at the end of this section.

According to the definition of LM–decomposable processes, two partitioning schemes are possible. For the definition of these schemes we have to introduce some notations.

Definition 3.2 *The set of all action names $\mathcal{Y}_1(P)$ that correspond to actions a process P may engage in is defined as:*

$$\begin{array}{lcllcl}
\mathcal{Y}_1(0) & = & \emptyset & \mathcal{Y}_1(P \parallel_S Q) & = & \mathcal{Y}_1(P) \cup \mathcal{Y}_1(Q) \\
\mathcal{Y}_1((a,\lambda).P) & = & \{a\} \cup \mathcal{Y}_1(P) & \mathcal{Y}_1(rec\ X : P) & = & \mathcal{Y}_1(P) \\
\mathcal{Y}_1(P\ +\ Q) & = & \mathcal{Y}_1(P) \cup \mathcal{Y}_1(Q) & \mathcal{Y}_1(P \backslash a) & = & \mathcal{Y}_1(P) \backslash \{a\}
\end{array}$$

Please note that the set of actions a process P will actually engage in is a subset of $\mathcal{Y}_1(P)$ as the synchronization may disable the execution of some actions.

Definition 3.3 *Let $P = L \parallel_S M$ be a LM–decomposable process. We define the set of all actions the process L executes independently from the process M as*

$$U := \mathcal{Y}_1(L) \backslash S \tag{16}$$

Definition 3.4 *Let $D \subseteq Act$, $P, Q \in \mathcal{L}$*

$$\begin{aligned} P \sim_D Q \quad :\leftrightarrow \quad & P = Q \\ \vee \quad & (\forall a \in Act \backslash D)(P \xrightarrow{a,,} P' \Rightarrow P' \sim_D Q \quad \vee \quad Q \xrightarrow{a,,} Q' \Rightarrow P \sim_D Q') \end{aligned} \tag{17}$$

Proposition 3.1 *The relation $\sim_D$ is an equivalence relation for a given set $D \subseteq Act$.*

Proof: Follows immediately from the definition. □

The equivalence relation $\sim_D$ judges two processes as equivalent if they can reach each other only with actions that are not in D. Now, it is easy to define two partitioning schemes for LM–decomposable processes.

Definition 3.5

$$PS_1(P) := \{Q/\sim_U \mid \ Q \ \textit{is reachable from} P\} \tag{18}$$

$$PS_2(P) := \{Q/\sim_{\overline{U}} \mid \ Q \ \textit{is reachable from} P\} \tag{19}$$

Obviously, the equivalence relation $\sim_D$ can be used to define partitions on the state space of more general processes, too. However, the selection of appropriate action sets D is not trivial in general. One possibility for a more general class of decomposable processes is shown next:

Definition 3.6 *A process $P \in \mathcal{L}$ with finite state space is* FR–decomposable, *if*

$$\exists F, R \subseteq Act \ \textit{ where actions in } F, R \textit{ have typically low transition rates}$$

The actions sets F and R should be interpretable as sets of actions that cause failures or carry out repairs respectively.

One natural partitioning scheme that can be implied for FR–decomposable processes is the partitioning scheme defined by the equivalence classes of the relation $\sim_{F \cup R}$.

Definition 3.7 *A decomposable process term is called Near Completely Decomposable (NCDP), if there is a set of actions S so that the equivalence relation $\sim_S$ leads to a partitioning scheme where the transition rates between different partitions are very low compared to other transition rates of the same model.*

The last definition is motivated by the well known *Near Complete Decomposability* property of some Markov chains and by the fact that performability models very often belong to this class. A process term that belongs to the NCDP class yields by definition a NCD Markov chain and is therefore especially suitable for aggregation techniques. Processes that are decomposable need only be analyzed by efficient blockwise SOR or JOR iteration schemes or by combinations of aggregation–disaggregation with iterative methods. Of course, decomposable processes are also candidates for transient analysis based on aggregation. In the next section an example is presented that falls in this NCDP class.

4 Example

To demonstrate the applicability of stochastic process algebras to failure modelling we present an extensive case study in this section. The model to be considered represents a multiprocesser mainframe that serves two purposes: on the one side it has to maintain a database and therefore has to process transactions submitted by a number of users, on the other side it is used for program development and has to provide computing capacity to programmers for compiling and testing their programs. As hardware failures occur relatively seldom in comparison to software failures, database inconsistencies, or operating system panics, we will focus on the latter kind of failures and neglect hardware failures.

One of the main advantages of process algebras is that it allows the creation of highly modular model descriptions. In our case, we can consider our system as the parallel composition of two components or processes (cf. Fig. 2).

$$System := Load \parallel_A Machine$$

where

$$A := \{user\_job, prog\_job, fail\}$$

The process $Load$ represents the system load caused by the database users, the programmers, and the various failures. The mainframe itself is modelled by the $Machine$ process.

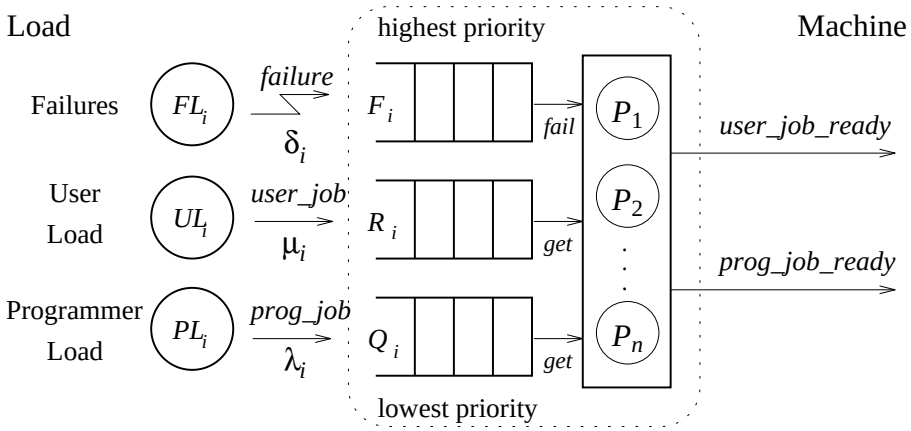

Figure 2: Model Structure

4.1 Load Modelling

As we want to examine the long term system behaviour dependent on the failure rates and the recovery strategy, we cannot model the various loads as Poisson processes. In a typical application like this the load is variable over time. Therefore, we model the three arrival streams by an Markov Modulated Poisson Process (MMPP).

$$Load := ProgLoad_1 \parallel_{\{c\}} UserLoad_1 \parallel_{\{c\}} FailLoad_1$$

$$
\begin{array}{lcll}
UserLoad_1 & := & (user\_job, \mu_1).UserLoad_1 & + \quad (c, \phi).UserLoad_2 \\
UserLoad_2 & := & (user\_job, \mu_2).UserLoad_2 & + \quad (c, \phi).UserLoad_3 \\
UserLoad_3 & := & (c, \phi).UserLoad_1 & \\
 & \vdots & & \vdots
\end{array}
$$

The processes $UserLoad_i$ and $ProgLoad_i$ generate repeatedly after an exponentally distributed time delay user jobs and programmer jobs with the rates μ_i and λ_i respectively. By analogy, the process $FailLoad_i$ generates with rate δ_i failures, which can be regarded as special jobs that require all processors and possess preemptive priority over the other job classes. All load processes change their phase after an exponentially–ϕ distributed delay. In phase 3 all load processes are idle. We assume

$$\lambda_1 = 0.01667 \text{ min}^{-1} \quad \lambda_2 = 0.16 \text{ min}^{-1} \quad \mu_1 = 0.033 \text{ min}^{-1} \quad \mu_2 = 2 \text{ min}^{-1}$$

$$\delta_1 = 0.00035 \text{ min}^{-1} \quad \delta_2 = 0.0007 \text{ min}^{-1} \quad \phi = 0.00334 \text{ min}^{-1}$$

which means that the mean duration of one phase is 5 hours and that in the second phase the programmer jobs and the failures occur two times more frequently than in phase one.

If we take a closer look at the load process, we may observe that it fulfills exactly the condition (15) because the process $Load$ can change its state only by executing action c, which however is not synchronized with the machine model. Therefore, this model falls in the class of LM–decomposable processes that were introduced in section 3.3.1. Moreover, this model represents also a FR–decomposable process with $F = \{fail\}$ and $R = \{repair\}$. As the transition rates of those actions are relatively low, this process belongs to the NCDP class.

4.2 The Machine Model

The second main component of our model represents the mainframe itself. First, we have to introduce a queue for jobs that have to wait until a processor becomes free. The processing unit is modelled by the parallel composition of 4 identical processes, each modelling one processor.

$$
\begin{array}{lcl}
Machine & := & Queues \parallel_B (P \parallel_{\{fail,repair} P \parallel_{\{fail,repair\}} P \parallel_{\{fail,repair\}} P) \\
Queues & := & F_0 \parallel_C (R_0 \parallel_{\{get\_prog\_job\}} Q_0)
\end{array}
$$

where

$$
\begin{array}{lcl}
B & = & \{get\_user\_job, get\_prog\_job, fail, repair\} \\
C & = & \{user\_job, prog\_job, get\_user\_job, get\_prog\_job\}
\end{array}
$$

4.2.1 The Queues

The queuing component is responsible for the buffering of incoming jobs. Additionally, we integrated a scheduling mechanism in this component. Jobs are processed according to a FIFO strategy with priorities. Programmer jobs have the lowest priority, while failures have the highest priority. To each class of jobs a separate queue process is assigned. The queue Q_0 stores the low priority jobs while the queues R_0 and F_0 are for the user jobs and the failures respectively.

$$
\begin{array}{lcll}
Q_0 & := & (prog\_job, \mathbb{1}).Q_1 & \\
Q_i & := & (prog\_job, \mathbb{1}).Q_{i+1} & + \quad (get\_prog\_job, \alpha).Q_{i-1} \\
Q_l & := & (get\_prog\_job, \alpha).Q_{l-1} & \\
R_0 & := & (user\_job, \mathbb{1}).R_1 & + \quad (get\_prog\_job, \mathbb{1}).R_0 \\
R_i & := & (user\_job, \mathbb{1}).R_{i+1} & + \quad (get\_user\_job, \alpha).R_{i-1} \\
R_l & := & (get\_user\_job, \alpha).R_{l-1} & \\
F_0 & := & (fail, \mathbb{1}).F_1 & \\
 & + & (get\_user\_job, \mathbb{1}).F_0 & + \quad (get\_prog\_job, \mathbb{1}).F_0 \\
 & + & (user\_job, \mathbb{1}).F_0 & + \quad (prog\_job, \mathbb{1}).F_0 \\
F_1 & := & (repair, \mathbb{1}).F_0 &
\end{array}
$$

The priority mechanism is realized by appropriate synchronization of the three queue processes. The process Q_i for instance is only able to deliver a job to a processor if the other two queues are in state R_0 and F_0. Otherwise, the action get_prog_job is not enabled. Another important fact is that the process F_i prohibits the insertion of new jobs if it is in state F_1, i.e. if a failure occured. The mean duration of the actions get_user_job and get_prog_job is $1/\alpha$ where $\alpha = 48$ min^{-1}.

4.2.2 The Processing Unit

The system under investigation contains 4 processors. Each processor waits until it can carry out a get action or until a failure occurs. As failures have preemptive priorities over the other two task classes, all processors stop processing if the $fail$ action becomes enabled and have to wait until the system will recover.

$$
\begin{array}{lcll}
P & := & (get\_user\_job, \mathbb{1}).PW_0 & + \quad (get\_prog\_job, \mathbb{1}).PW_1 \; + \; (fail, \mathbb{1}).P_f \\
PW_0 & := & (user\_job\_ready, \nu).P & + \quad (fail, \mathbb{1}).P_f \\
PW_1 & := & (prog\_job\_ready, \xi).P & + \quad (fail, \mathbb{1}).P_f \\
P_f & := & (repair, \beta/4).P &
\end{array}
$$

The following values for the service time distribution and the repair time distribution were used:

$$\beta = 0.01 \text{ min}^{-1} \quad \nu = 12 \text{ min}^{-1} \quad \xi = 0.3 \text{ min}^{-1}$$

There are many ways to describe the machine model. This applies to both the queuing component and the processing component. Questions of some importance are whether to model the queuing component by one monolithic process or to use one queue for each class of jobs. By analogy, the question whether to use one single process for the description of the processors or to use the parallel composition of several identical processes, each modelling one processor, plays an important role for model and solution complexity.

For the sake of simplicity, we present here the second solution although in practice we modelled the processing unit as one single process in order to reduce the state space size and to save computation time. Obviously, this makes models more complex and one main advantage of the process algebra approach, namely *compositionality*, seems to disappear. However, it should be mentioned that the algebraic theory behind process algebras like TIPP allows the replacement of components of one process by equivalent ones without changing the behaviour of the model [11, 20]. Moreover,

the existence of a sound axiomatization allows this step to be carried out automatically [21]. In Hillston's thesis this was demonstrated by an example similar to this. Additionally, the relationship to the concept of *lumpability* was examined [22].

4.3 Model Evaluation

Two important aspects that are of some interest considering performability models are: how much does the occurrence of failures influence the performance of the system and how well can different error recovery strategies help in increasing the system availability. To investigate those aspects for the current example, we applied steady state analysis in order to obtain some characteristic performance measures and dependability measures.

First, we have to allocate the length of the queues Q_i and R_i in a way that the blocking probability is sufficiently small. In the first analyzed model we assigned 40 places to the first queue and 10 places to the second one. The resulting state space contained 21648 states and the numerical solution took about 165 seconds using Gauß–Seidel's method. However, a closer look at the results showed that the average queue lengths are sufficiently small that reducing the queue lengths to 10 and 4 would not sacrifice model accuracy very much. The reduced model contains only 2640 states and solution takes now less than 5 seconds. Fig. 3 and 4 show these results.

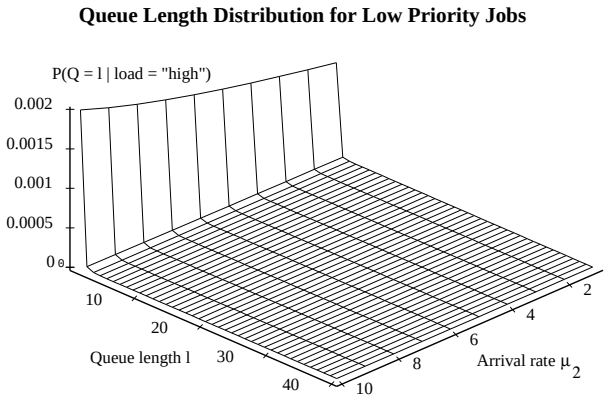

Figure 3:

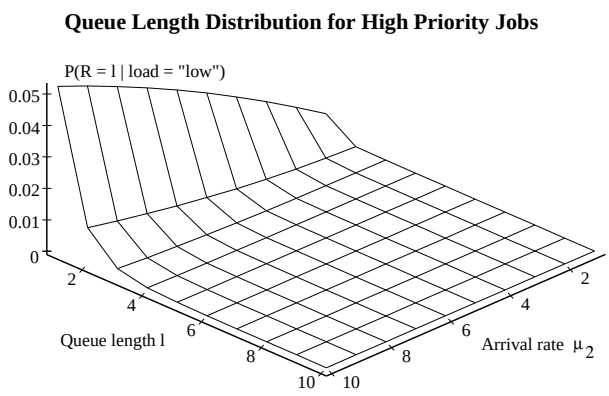

Figure 4:

The impact of various repair strategies and failure arrival rates is another very interesting question in such models. Therefore, we varied the repair rate β and the failure rate during high load (δ_2) and computed the mean availability (cf. Fig. 5) as well as the throughput (cf. Fig. 6) using appropriate reward functions (cf. section 3.2).

$$A(\infty) = \lim_{t \to \infty} A(t) \qquad M(\infty) = \lim_{t \to \infty} M(t)$$

As we can see, there is a strong correlation between both measures. Increasing the repair rate leads to a better throughput of high priority jobs as well as an improvement of system availability.

Using steady state analysis a number of interesting measures can be derived that give useful information about the system performance and dependability. However, questions regarding the time dependent behaviour like: how does the system behave on overload situations or how fast does it reach a steady state remain unanswered. Here, transient analysis is an important tool to

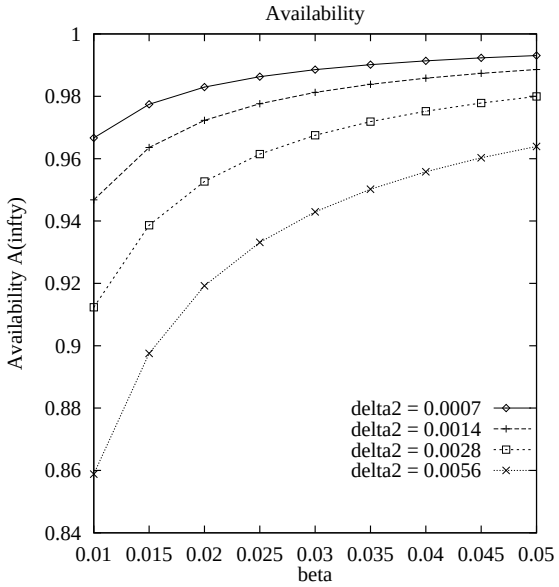

Figure 5:

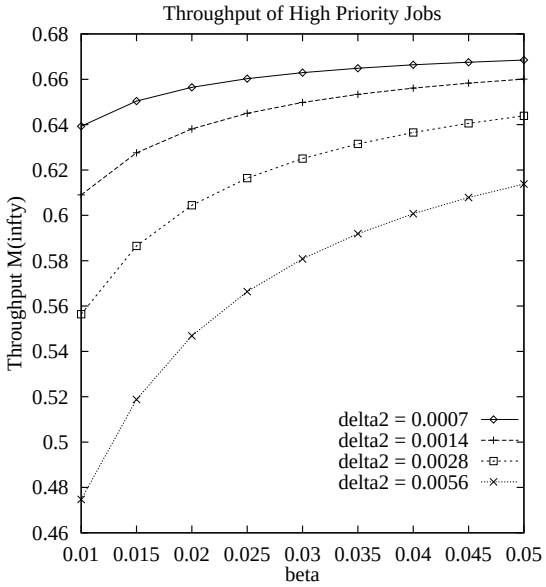

Figure 6:

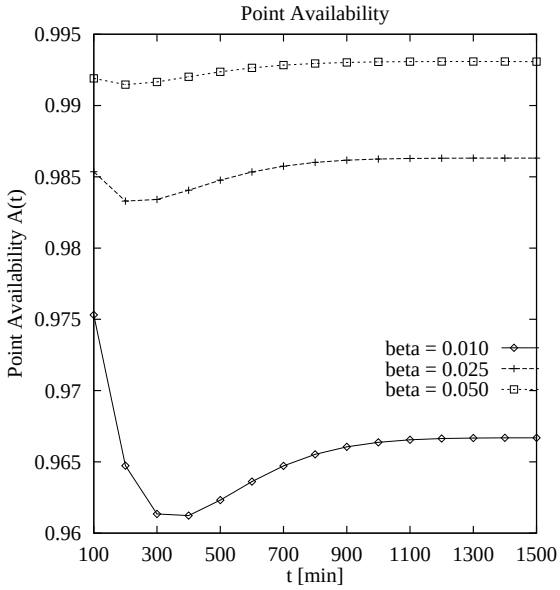

Figure 7:

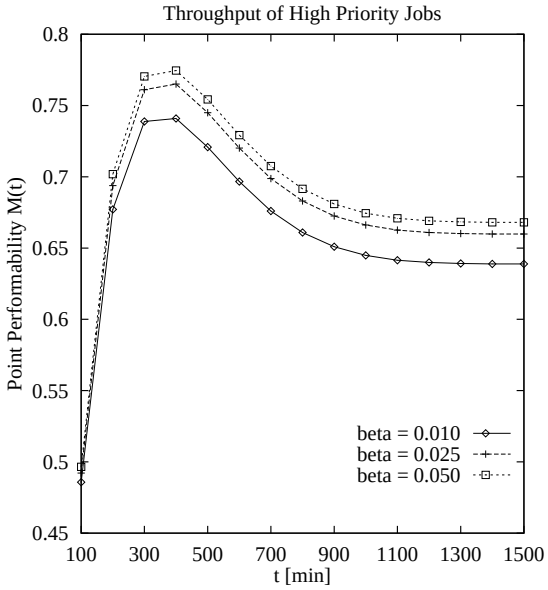

Figure 8:

predict such values. For our example we examined the transient behaviour for a duration of 1500 minutes using the refined randomization technique introduced by [27]. Fig. 7 shows how the point availability changes over the time dependent on three different repair rates β. Fig. 8 shows a point performability measure.

For $\beta = 0.01$ we can see clearly how the phase change in the load process affects the point availability. After $t = 1400$ minutes the system reaches its steady state.

Finally, we changed the initial phases of the three load processes in order to analyze how this interacts with the system's performability. Obviously, the initial phase does not influence the steady state behaviour as long as all the three load processes are in the same phase. However, if we look at the time dependent measures, things look a bit different (Fig. 9 and Fig. 10).

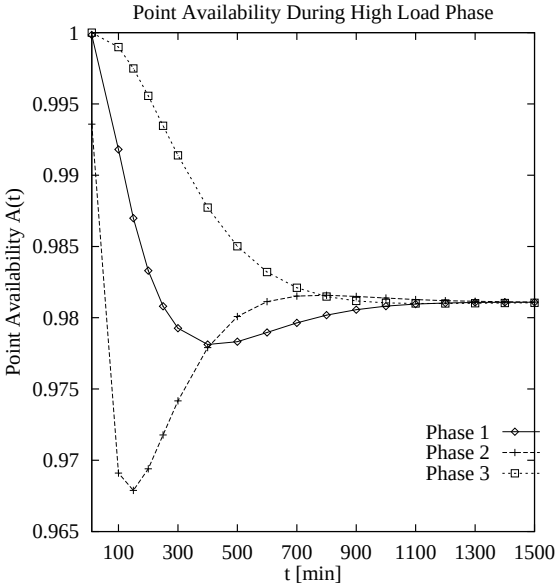

Figure 9:

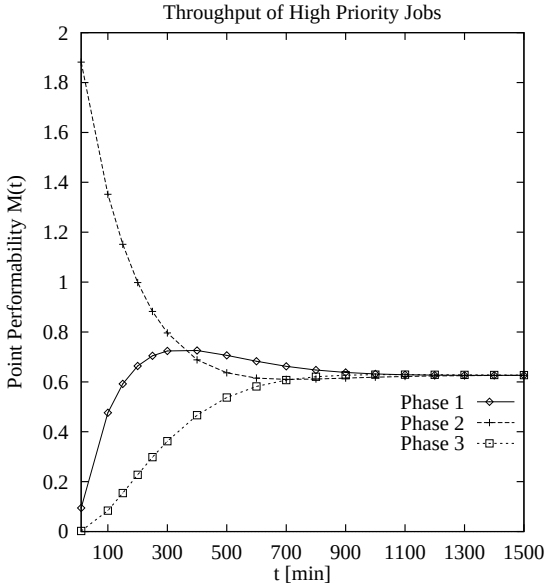

Figure 10:

We observe that for a long period of time before the steady state is reached the system's point availability is better when the system is started in phase 3, i.e. in the silent phase (Fig. 9). Fig. 10 reflects clearly the phase changes in the load processes dependent on the initial phase.

5 Conclusion and Prospects

Performability analysis of computer systems is a very difficult task that requires usually a very deeply technical knowledge about Markov processes as well as a lot of experience. Using high level specification techniques, such as stochastic process algebras, Petri nets, or automata networks, it is possible to automate this process. Consequently, performability analysis can be taught to a wider audience. Moreover, it is possible to integrate this task into a more comprehensive methodology, like the software development lifecycle of communication systems. Stochastic process algebras

and automata networks are especially suitable for this because of their close relationship to the widely accepted formal description techniques LOTOS, SDL, and ESTELLE.

One of the main drawbacks when using high level modelling techniques for performability analysis is the lack of tools that provide analysis algorithms tailored to the needs of performability models. The exploitation of the structural information contained in each model in order to guide the choice of appropriate algorithms seems to be the right way to solve this problem and this was already demonstrated for stochastic Petri nets in [1]. We showed how to adopt this idea for stochastic process algebras by presenting a relatively general class of processes where the use of efficient solution algorithms is feasible.

Another disadvantage is that high level specifications are very often too detailed and can therefore increase the computation effort dramatically. This problem has been known since the beginnings of elementary queueing theory and it is usually tackled by chosing a compact representation of the state space. This is usually done by hand and requires a lot of experience. Current research in the area of stochastic process algebras aims at providing facilities to automate this process and first interesting results were presented in [6], [22].

As far as the topics covered in this paper are concerned, there are many open issues that need to be further investigated. First, we have to evaluate the comprehensiveness of the introduced class of decomposable processes by studying more case studies. Further, we plan to integrate several aggregation–disaggregation techniques in our prototype evaluation tool. The main challenge, however, will be to show the applicability and the effectiveness of iterative or multilevel aggregation–disaggregation schemes for the analysis of decomposable processes or more general processes.

Acknowledgement

The authors would like to thank S. Gilmore, H. Hermanns, and J. Hillston for proof reading early versions of this paper and for many valuable discussions.

References

[1] H. H. Ammar and S. M. Rezaul Islam. Time Scale Decomposition of a Class of Generalized Stochastic Petri Net Models. *IEEE Transactions on Software Engineering,* 15(6):809–820, June 1989.

[2] M. Beaudry. Performance Related Reliability for Computer Systems. *IEEE Transactions on Computers,* C(27):540–547, June 1978.

[3] A. Bobbio and K.S. Trivedi. An Aggregation Technique for the Transient Analysis of Stiff Markov Chains. *IEEE Transactions on Computers,* C-35(9):803–814, September 1986.

[4] A. Bobbio and K.S. Trivedi. Computing Cumulative Measures of Stiff Markov Chains Using Aggregation. *IEEE Transactions on Computers,* 39(5):1291–1298, October 1990.

[5] P. Buchholz. Numerical Solution Methods Based on Structured Descriptions of Markovian Models. In G. Balbo and G. Serazzi, editors, *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation,* pages 242–258. Elsevier Science Publisher B.V., 1992.

[6] P. Buchholz. On a Markovian Process Algebra. Forschungsbericht 500/1994, Informatik IV, Universität Dortmund, Postfach 500 500, 4600 Dortmund 50, Germany, 1994.

[7] R. Chakka and I. Mitrani. A Numerical Solution Method for Multiprocessor Systems with General Breakdowns and Repairs. In R.J. Pooley and J. Hillston, editors, *Proceedings of the 6th International Conference on Modelling Techniques and Tools*, pages 289–304, Edinburgh, September 1992.

[8] G. Ciardo, J. Muppala, and K.S. Trivedi. On the Solution of GSPN Reward Models. *Performance Evaluation*, 12(4):237–254, 1991.

[9] P.J. Courtois. *Decomposability: Queueing and Computer System Applications*. Academic Press, New York, 1977.

[10] H. de Meer. *Transient Performance Evaluation and Optimization of Reconfigurable Fault–Tolerant Computer Systems (in German)*. PhD thesis, University of Erlangen, 1992.

[11] Huu Trung Do. Entwurf von Prozeßsprachen zur Leistungsbewertung. Master's thesis, Universität Erlangen–Nürnberg, IMMD VII, Juli 1993.

[12] B.L. Fox and P.W. Glynn. Computing Poisson Probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[13] H.R. Gail and E. de Souza e Silva. Performability Analysis of Computer Systems: From Model Specification to Solution. *Performance Evaluation*, 14:157–196, 1992.

[14] N. Götz. *Stochastic Process Algebras – Integration of Functional Design and Performance Evaluation of Distributed Systems (in German)*. PhD thesis, University of Erlangen, 1994.

[15] Norbert Götz, Ulrich Herzog, and Michael Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Proc. of the 16th Int. Symposium on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE '93*. Springer, 1993. LNCS 729.

[16] W. Grassmann. Transient Solutions of Markovian Queues. *Europ. J. Oper. Res*, 1:396–402, 1977.

[17] B. Haverkort, J. Muppala, S. Woolet, and K. Trivedi. Composite Performance and Dependability Analysis. *Performance Evaluation*, 14:197–215, 1992.

[18] B. Haverkort, N.M. van Dijk, and I.G. Niemegeers (editors). Special Issue on Performability Modelling of Computer and Communication Systems. *Performance Evaluation*, 14(3-4), February 1992.

[19] B.R. Haverkort and K.S. Trivedi. Specification Techniques for Markov Reward Models. *Discrete Event Systems: Theory and Applications*, 3:219–247, 1993.

[20] H. Hermanns. Semantik für Prozeßsprachen zur Leistungsbewertung. Master's thesis, Universität Erlangen–Nürnberg, IMMD VII, November 1993.

[21] H. Hermanns and M. Rettelbach. Markovian Processes go Algebra. Technical Report 10/94, IMMD VII, Friedrich-Alexander-Univeristät, Erlangen-Nürnberg, Germany, 1994.

[22] Jane Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.

[23] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[24] G. Horton and S. Leutenegger. A Multi-Level Solution Algorithm for Steady-State Markov Chains. *ACM Performance Evaluation Review*, 22(1):191–200, May 1994. Proceedings of the ACM Sigmetrics and Performance 1994, International Conference on Measurement and Modeling of Computer Systems.

[25] L. Kleinrock. *Queueing Systems*, volume 1: Theory. John Wiley & Sons, 1975.

[26] U.R. Krieger, B. Müller-Clostermann, and M. Sczittnick. Modeling and Analysis of Communication Systems Based on Computational Methods for Markov Chains. *IEEE Journal on Selected Areas in Communications*, 8(9):1630–1648, 1990.

[27] C. Lindemann. Employing the Randomization Technique for Solving Stochastic Petri Net Models. In A. Lehmann and F. Lehmann, editors, *Proc. 6. GI/ITG Conf. on Modelling, Measurement and Evaluation of Computing Systems (MMB '91)*, pages 306–319, Munich, Germany, 1991. Springer.

[28] J.F. Meyer. On Evaluating the Performability of Degradable Computing Systems. *IEEE Transactions on Computers*, C-29(8):720–731, August 1980.

[29] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[30] P. Naor and U. Yechialy. Queuing Problems with Heterogeneous Arrivals and Service. *Operations Research*, 19:722–734, 1971.

[31] M.F. Neuts. *Matrix–Geometric Solutions in Stochastic Models.* The John Hopkins University Press, Baltimore, MD, 1981.

[32] B.D. Plateau and S.K. Tripathi. Performance Analysis of Synchronisation for Two Communicating Processes. *Performance Evaluation*, 8:35–320, 1988.

[33] M. Rettelbach and M. Siegle. Deriving Lumped Markov Chains from SPA Descriptions. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, Regensberg, Germany, July 1994.

[34] I. Schieferdecker and A. Wolisz. The Timed Interacting Systems Approach. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, Regensberg, Germany, July 1994.

[35] P.J. Schweitzer. Aggregation Methods for Large Markov Chains. In G. Iazeolla, P.J. Courtois, and A. Hordijk, editors, *Mathematical Computer Performance and Reliability*, pages 275–285. North Holland, 1984.

[36] M. Siegle. Reduced Markov Models of Parallel Programs with Replicated Processes. In *2nd EUROMICRO Workshop on "Parallel and Distributed Processing"*, pages 126–133, Malaga, Spain, January 1994.

[37] W.J. Steward. Recursive Procedures for the Numerical Solution of Markov Chains. In H.G. Perros and T. Altiok, editors, *Queueing Networks with Blocking*, pages 229–247. Elsevier Science Publisher B.V. (North-Holland), 1989.

[38] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, 1982.

[39] K.S. Trivedi and M. Malhotra. Reliability and Performability Techniques and Tools: A Survey. In B. Walke and O. Spaniol, editors, *Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen*, pages 27–48, Aachen, September 1993. Springer.

[40] H.C. White and L.S. Christie. Queuing with Preemptive Priorities or Breakdown. *Operations Research*, 6:79–95, 1958.