

Balls and buckets for discrete time Markov chain generation [†]

Héctor Cancela Bosi

cancela@fing.edu.uy

Investigación Operativa - InCo - Pedeciba Informática

Facultad de Ingeniería, Universidad de the República

Montevideo, Uruguay

Abstract

*This work presents an extension of the balls and buckets description formalism to the case of discrete time systems. This formalism allows for flexible modeling and compact representation of Markovian systems, supporting automatic generation of both the state space and the transition matrix of the underlying canonical Markov chain. A discrete time balls and buckets interpreter has been implemented in the **BB** library, which provides different basic services (Markov chain generation, storage, exact evaluation, and simulation) supporting system description and performance and dependability quantitative analysis.*

As an application example, we present the balls and buckets model and the performance analysis of an ATM switch architecture, which is of interest in high performance network design. By using simulation techniques, we reproduce results (already established by other methods in the literature) showing that, given a fixed load, the performance of this kind of switches is highly sensitive to the characteristics of the data packets arrival process.

Keywords: *system modeling, simulation, performance and dependability evaluation, distributed systems and computer networks.*

1 Introduction

The complexity of the new generations of computer and communications systems and the fact that many activities depend on their good performance has promoted a growing attention to modeling and evaluation techniques applicable to these systems. The main properties are the *performance* and the *dependability*, this last being of particular interest in so-called critical systems, for example systems controlling nuclear generators, commercial, military or spatial flight systems, etc. Other interesting measures include the *reliability*, the *availability*, the *performability* (performance-dependability combined measure), etc. To study these measures we model the systems using stochastic processes, typically in a Markovian setting. Many numerical techniques have been developed to evaluate Markovian processes. The main problem is that the state space size of the process grows

[†]This work was supported by grants from Action U93E03 of ECOS - Scientific Cooperation between France and Uruguay Program, and from CONICYT (Uruguay)

(often very fast) with the complexity and detail level of the model, which makes it impossible for the user to directly specify the corresponding process. This job requires a systematic modeling approach, with the use of high level interfaces offering a compact representation of the studied system.

Within the Markovian setting, we find two main model families, which correspond to the use of a continuous or discrete time parameter. In the continuous time setting, there are a number of tools based on formalisms which allow to give a compact representation of the system studied, and generate afterwards the underlying Markov process. We can find, among others, tools based on Petri nets [3, 10, 11, 13, 19], stochastic activity networks [17, 19, 20], queuing systems [16], *balls and buckets* [8, 21], Markovian event systems [14], automata networks [18], process algebras [15], logical frameworks [9], hybrid logical-object oriented approaches [4, 6], and specific “machine-repairmen”-like languages as SAVE [5].

The discrete time setting has been less explored, but has also been getting more attention recently, in response to recent changes in the communication networks technology. In particular, it is necessary to employ a discrete time parameter in order to model more precisely both software and hardware behavior. A case of particular interest is the modeling of high performance communication networks based on ATM. This need has motivated recent works presenting discrete time formalisms (discrete time automata networks [18], discrete time stochastic Petri nets [12], discrete time queuing networks [24]), which can be used to study the behavior of those systems. Nevertheless, most existing modeling tools restrict themselves to only continuous time formalisms, and do not include discrete time models.

In this work we present an extension of the *balls and buckets* formalism (previously employed for continuous time system modeling) to the discrete time case. This formalism can be used to give system descriptions more compact than those based on Markov chains, while maintaining the same expressive power (the classes of systems that can be represented are equivalent). To extend the formalism to the discrete time case, it is necessary to expand its semantics, to take into account the fact that many changes of the system state can take place simultaneously, at the same time unit. This extension has been incorporated into the **BB** model generation library [7, 8], which previously only supported continuous time models. The library is composed by a set of classes and methods for *balls and buckets* model construction and evaluation. In particular, it is possible to find the Markov chain equivalent to any specific **BB** model, opening access to the battery of existing analytical methods; also it is possible to use simulation methods, when the size of the model makes impractical (or impossible) the first alternative. As an example, we have taken a discrete time model of resource contention for a specific ATM switch architecture.

This paper is organized as follows. Next section presents the discrete time *balls and buckets* formalism. In Section 3, we discuss the implementation of this formalism in the **BB** library. In Section 4 we develop as an application example the modeling and performance evaluation of an ATM switch. Finally, some conclusions and future work perspectives are given in Section 5.

2 Discrete time *balls and buckets* formalism

2.1 Informal presentation and example

The *balls and buckets* modeling formalism has been proposed by Stewart [21] as a compact means for representing continuous time Markovian systems by the use of a reduced set of simple rules, allowing for flexible modeling and automatic state space and infinitesimal generator matrix construction.

In its original form it is not suitable to discrete time system modeling. In order to cope with this case, we have developed an extended version of the formalism. While preserving the main philosophy of the previous approach, it has been necessary to modify and widen the formalism semantics, specially because it becomes necessary to explicitly take into account the possibility that many system state modifications may take pace in a single discrete time unit. The differences between the continuous and the discrete time versions are enough to preclude an easy unified presentation, so from now on we will refer exclusively to the new discrete time formalism.

In the discrete time formalism we propose, the system state is represented by a set of L buckets, each containing a number of balls $n[i]$ (with $n[i] \in \{0, \dots, nmax[i]\}$, $i = 1, 2, \dots, L$). The balls can move among the different buckets. The movement of a ball from a bucket i to a bucket j is called a *transfer* from *source* i to *destination* j , and can happen only if $(i, j) \in \mathcal{L}$ (the transfer belongs to the list of admissible transfers), and also $n[i] > 0$ and $n[j] < nmax[j]$, or $n[j] = nmax[j]$ if $i = j$ (enabling condition). As a result of a transfer from i to j , the number of balls $n[j]$ goes up by 1 unit and $n[i]$ goes down also by 1. In a given time unit, a bucket may be the source of at most one transfer, but it may be the destination of many. The transfers happen with given probabilities, which may depend upon the global system state. The sum of the probabilities of transfers having the same source bucket must be less or equal to one. A transfer may be followed by an arbitrary modification of the buckets contents, which takes place instantaneously. Since at the same time many transfers (having different source buckets) may happen, we consider that these transfers take place ordered by their input bucket position, transfers having lower numbered input buckets taking place before higher numbered buckets (this is the same as giving arbitrary priorities to the buckets, and re-ordering them). In this case a transfer is possible if the enabling condition was true at the cycle start and if it is still true after consideration of the effects of all transfers having greater priority (i.e, having smaller input bucket number).

A *transition* is a system state change, resulting from the composition of all transfers and instantaneous modifications which take place in a single time unit.

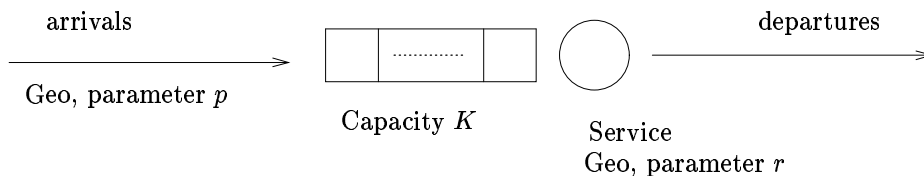


Figure 1: Geo/Geo/1/K queuing system

As an example let us consider a Geo/Geo/1/K discrete time queuing system, which is shown in Figure 1. In each time unit a new customer can arrive with probability p (the arrival law is a Bernoulli process with parameter p). The service time is a random variable with geometric distribution and parameter r (we will denote $\bar{p} = 1 - p$, and $\bar{r} = 1 - r$). The queue has a fixed capacity of K customers. We suppose that in this system “arrivals see departures”, that’s to say during the same time unit departures from the system are processed before arrivals.

The following parameters define (informally) a *balls and buckets* model for this system:

- $L = 2$: two buckets, the first one corresponding to the customers in the queue (including the one being serviced, if any) and the second one corresponding to the system environment;

- $nmax[1] = K + 1;$
 $nmax[2] = K + 1;$
- $\mathcal{L} = \{(2, 1), (1, 2)\}$: the admissible transfers;
- transfer modeling a customer arrival: from bucket 2 to bucket 1, with probability p ;
- transfer modeling the end of a service at the station:
 from bucket 1 to bucket 2, with probability r ;
- initial state: $(0, K + 1)$;
- no instantaneous modifications.

In Figure 2, we can see the Markov chain which corresponds to this system.

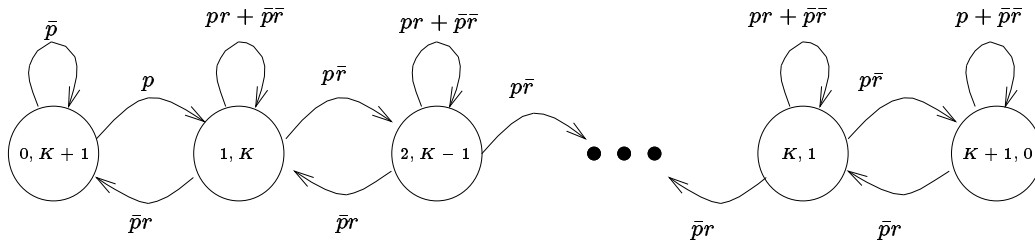


Figure 2: Markov chain corresponding to the Geo/Geo/1/K queue

2.2 Formal Definition

A discrete time *balls and buckets* model is defined by the 6-uple $(L, \overrightarrow{nmax}, \vec{e}_0, \mathcal{L}, \lambda, \Gamma)$, where:

- $L \geq 1$ is the number of buckets;
- $\overrightarrow{nmax} \in \mathbb{N}^L$ is the maximum number of balls in each bucket;
- $\vec{e}_0 \in \mathbb{N}^L$, $0 \leq e_0[i] \leq nmax[i] \forall i \in \{1, \dots, L\}$, is the initial state;
- $\mathcal{L} \subseteq L^2$ is the list of admissible transfers;
- $\lambda \in (\mathbb{N}^L \rightarrow [0, 1])^{\mathcal{L}}$ is the transfer probability function (as function of the system state). For each \vec{n} , i, j , such that $(i, j) \in \mathcal{L}$, $0 < n[i]$ and $n[j] < nmax[j]$ (or $n[j] = nmax[j]$ if $i = j$),

$$\lambda[i, j](\vec{n}) = \text{transfer probability of a ball from bucket } i \text{ to bucket } j;$$

If $(i, j) \notin \mathcal{L}$, $n[i] = 0$ or if $n[j] = nmax[j]$, by definition $\lambda[i, j](\vec{n}) = 0$. Also we always have $\sum_j \lambda[i, j](\vec{n}) \leq 1, \forall i, \forall \vec{n}$.

- $\Gamma \in (\mathbb{N}^L \rightarrow (\mathbb{N}^L \times [0, 1])^*)^{\mathcal{L}}$ is the instantaneous modifications function; for each \vec{n} , resulting from a transfer from i to j , Γ gives the set of all possible instantaneous modifications:

$$\Gamma[i, j](\vec{n}) = \begin{cases} (\vec{n}, 1) & \text{if there is no instantaneous modification;} \\ \left\{ (\vec{m}_1, p_1), \dots, (\vec{m}_H, p_H), (\vec{n}, 1 - \sum_{h=1}^H p_h) \right\} & \text{where } \vec{m}_1, \dots, \vec{m}_H \text{ are the accessible states} \\ & \text{from state } \vec{n} \text{ (after a transfer from bucket } i \\ & \text{to bucket } j). \text{ The probability that the transition} \\ & \text{to } \vec{m}_h \text{ is chosen is } p_h. \end{cases}$$

It is easy to show that discrete time balls and buckets models are equivalent to homogeneous discrete time Markov chains with finite state space. If we generalize the *balls and buckets* formalism allowing for the use of buckets with a non-bounded number of balls, we have the class of homogeneous discrete time Markov chains (with state space size possibly being infinite). In this case, it is not possible to generate the state space and the transition matrix, but it is possible to use techniques (such as simulation or bounding methods) that give approximations to useful system measures [1, 2, 22].

3 Implementation

3.1 Architecture of the BB library

The **BB** library, presented in [7] and [8], is a set of classes and methods implemented in C++ which, given a system dependability or performance model, build the underlying Markov process which can then be used to compute the desired measures. The library is divided into two main layers. The first layer (which we call the core of the library) is independent of the chosen modeling paradigm. The second layer, which we call the BB interpreter, is a specialization of the core for the treatment of *balls and buckets* models. These two layers give a number of services, which can be used by a third layer corresponding to user programs. In particular, it is possible to develop translators which can generate a *balls and buckets* model from a description of a system written in another formalism (as for example GSPNs).

The core of the library is composed by a set of generic classes and methods, which are independent from the modeling formalism. They provide basic abilities such as model generation, display, storage, and simulation.

At this level, a system is described by the function giving all possible transitions from any state, and their associated probabilities. Also the initial state should be known. Using this information, the core can build the set \mathcal{S} of accessible states and the equivalent Markov process X with state space \mathcal{S} and transition matrix M .

The implemented algorithm builds the process transition matrix incrementally, by a BFS search from the initial state. This technique has a number of advantages: for example, only states that can be reached from the initial state will be taken into account in the transition matrix (which is much more efficient than considering the Cartesian product of the system's individual states, which is the state space used in other modeling tools).

The second layer of the library contains all the necessary information to store and manipulate the states of a system modeled in terms of *balls and buckets*. This includes the implementation of virtual methods defined at the preceding layer. The user specifies the model as a 6-uple

$(L, \overline{nm\alpha x}, \vec{e}_0, \mathcal{L}, \lambda, \Gamma)$. The functions λ and Γ are given as C++ procedures, which allows a great specification flexibility. The system is represented as a *babSystem* class object, whose methods give transparent access to the library services.

3.2 Transition generation algorithm

The interaction of the two levels of the library is based on an algorithm which from a *balls and buckets* model and a state \vec{e} can give all transitions, with their respective probabilities. This algorithm takes the system description in terms of transfers and instantaneous modifications, and reconstructs its low level behavior.

In the **BB** library we have implemented such an algorithm, which can be seen in Figure 3, which stores in a list Δ the low level information built from the λ and Γ functions specified in the *balls and buckets* model.

To obtain all the combinations of transfers and instantaneous modifications, we consider in order all the buckets, and at each step we apply all the transfers (and corresponding instantaneous modifications) having as source the fixed bucket, over all previously generated states. In this way, we find the new accessible states (with their probability). We multiply the probability of the old states by their probability of not being the source of a transfer in the considered step. For the new step, we add the new states to the set Δ .

4 ATM Switch Model

4.1 Switch Architecture

The Asynchronous Transfer Mode (ATM) is a technology currently considered for high performance communication networks [23]. These networks must be capable of carrying a large number of highly bursty input links, such as voice, video, and large file transfer. The traffic in these networks is segmented in fixed size packets (or cells). The time is also slotted, each slot being equal to the time necessary to transmit a single cell. An important ATM component is the switch or multiplexer. There are three class of ATM switch architectures: shared-memory, shared-medium, and space-division.

In this section, we will study a shared-medium ATM switch presented in [25], which appears in Figure 4. In this model, all cells arriving from the input links are routed to output links over a common high speed medium, such as a parallel bus. There is some internal storage space (input and output buffers). If an input buffer is full, arriving cells will be lost; if an output buffer is full, there will be blocking at the routing level.

The following are the switch characteristics:

- **Input links:** there are N input links, each generating cells following an interrupted Bernoulli process (IBP) as follows: each input link has two states, Active and Passive. In a given time slot, a passive input link i can go to active state with probability $1 - q[i]$; and an active link can go to passive state with probability $1 - p[i]$. When an input link goes from passive to active state, it generates a cell; while an input link is active, it can generate a cell at each time slot with probability $\alpha[i]$.
- **Input buffers:** there are N input buffers of size M , each buffer being associated to an input links. If there is an arrival from input link i and the corresponding buffer is full, the cell is

```

Input: state  $\vec{e}$ 
        function  $\lambda$ 
        function  $\Gamma$ 
Output: list  $\Delta$ 
Begin
     $\Delta := \{(\vec{e}, 1)\}$ 
    Repeat for all buckets  $i$ 
         $\Delta_i := \emptyset$ 
        Repeat for all pairs  $(\vec{e}_k, p_k)$  in  $\Delta$ 
             $\Delta_{ik} := \emptyset$ 
            Repeat for all transfers  $i \rightarrow j$ 
                enabled at the same time in state  $\vec{e}$  and in state  $\vec{e}_k$ 
                 $p_{ij} := \lambda[i, j](\vec{e}_k)$ 
                If  $(p_{ij} > 0)$  then
                     $\vec{e}_k^* := \vec{e}_k$ 
                     $e_k^*[i]--$ 
                     $e_k^*[j]++$ 
                     $\Delta_{ikj} := \Gamma[i, j](\vec{e}_k^*)$ 
                    Repeat for all pairs  $(\vec{m}_l, p_l)$  in  $\Delta_{ikj}$ 
                         $p_l := p_l \times p_{ij} \times p_k$ 
                    End Repeat
                     $\Delta_{ik} := \Delta_{ik} \cup \Delta_{ikj}$ 
                End If
            End Repeat
             $p_k := p_k \times (1 - \sum_j p_{ij})$ 
             $\Delta_i := \Delta_i \cup \Delta_{ik}$ 
        End Repeat
     $\Delta := \Delta \cup \Delta_i$ 
End Repeat
Return  $\Delta$ 
End
    
```

Figure 3: Generation algorithm pseudocode

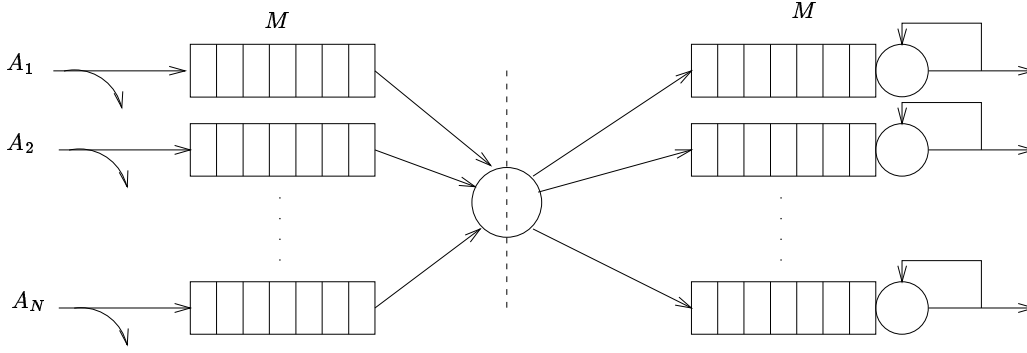


Figure 4: A shared-medium ATM switch architecture

lost.

- **Output buffers:** there are N output buffers of size M , each buffer being associated to an output link. The service time of each of the output buffers is a geometrical random variable of parameter $r[i]$ (at each time slot there is a probability $1 - r[i]$ that the cell being served leaves the system).
- **Switching:** there is a single bus for packet switching. The bus works at a speed N times faster than the input links. We will take a bus service policy of type TDM (Time Division Multiplexing), in which each bus slot i is preassigned to input buffer i ; in this way, it is possible to process a cell from each of the input buffers in a single (arrival) time slot. Each of the cells arriving in input link i has as destination the output line j with probability $D[i, j]$. If the cell being serviced at input buffer i has destination j and output buffer j is full, the cell will be blocked and will wait for its turn in the next cycle.

4.2 Performance evaluation

As input buffers only have a finite storage capacity, the cells arriving while they are full are lost. The loss probability P_L is one of the most important performance parameters in a packet switch.

For each input link, we can compute the loss probability from the steady-state distribution by the following formula:

$$P_L[i] = \frac{\sum_{\vec{n} \in F_P} Prob(\vec{n})(1 - q[i]) + \sum_{\vec{n} \in F_A} Prob(\vec{n})p[i]\alpha[i]}{\sum_{\vec{n} \in P} Prob(\vec{n})(1 - q[i]) + \sum_{\vec{n} \in A} Prob(\vec{n})p[i]\alpha[i]}$$

with $Prob(\vec{n})$ the steady-state probability of the system being in state \vec{n} , P the set of states where the input link is in passive state, A the set of states where the input link is in active state, F_P the set of states where the input link is in passive state and the input buffer is full, F_A the set of states where the input link is in active state and the input buffer is full.

The denominator is equal to the mean cell generation rate $\rho[i]$ of input link i , which verifies:

$$\rho[i] = \frac{\alpha[i](1 - q[i])}{2 - p[i] - q[i]}.$$

p	0.10	0.30	0.50	0.70	0.90
q	0.82	0.86	0.90	0.94	0.98
P_L	3.63×10^{-4}	9.00×10^{-4}	1.81×10^{-3}	3.01×10^{-3}	5.74×10^{-3}
Var(estimation)	2.54×10^{-9}	8.76×10^{-9}	2.72×10^{-8}	8.01×10^{-8}	3.61×10^{-7}

Table 1: Simulation results for the cell loss probability in an ATM switch

The numerator can be computed from the steady-state distribution if the size of the system is small, or by regenerative simulation.

We will study the effect of the traffic characteristics on the loss probability $P_L = \sum_{i=1}^N P_L[i]$, for a given cell generation rate. We will take the following parameters as fixed: $N = 4$, $\alpha[i] = 0.5 \forall i$, $r[i] = 0.1 \forall i$, $D[i, j] = 1/4 \forall i, j$, and we will consider the values of $p[i]$ from 0.1 to 0.9 changing at the same time the values of $q[i]$ from 0.82 to 0.98 so that the generation rate is constant ($\rho[i] = 1/12$). The small values of $p[i]$ correspond to network traffic such that the average cell burst length is small; big values of $p[i]$ model the case where the average cell burst is long.

Using the **BB** library, we developed a *balls and buckets* model for the switch. The simulation facilities included in the library were employed to compute the cell loss probability. The computing equipment was a Sun Ultra 30 Model 295 workstation, and the total time needed to compute 100.000 independent runs for each of the five parameter combinations simulated was less than two hours. Table 1 shows the estimated values for cell loss probability (obtained by simulation), as well as the estimation precision. We can appreciate how the loss probability goes up very rapidly as the average burst length grows, even if the cell arrival rate doesn't change. These phenomena are very important in ATM networks, and must be taken into account during the design phase, specially as traffic tends to be very bursty in the present-day telecommunications networks.

5 Conclusions

In this paper, we have presented an extension to the *balls and buckets* formalism allowing discrete time system modeling. This extension has been implemented in the **BB** library, giving access to the services it provides (Markov chain generation, storage, simulation).

As an application example, we have considered an ATM switch architecture and we have developed its *balls and buckets* model. This architecture performance was evaluated (for a number of different parameter values) by using the simulation primitives offered by the **BB** library; showing how the system performance, given a fixed load, depends on the traffic characteristics (if there are long bursts in the arrivals, the process autocorrelation is bigger, and performances diminish).

As future work, we can find on one hand the application of the **BB** library to other computer and communication systems evaluation. Another goal is to widen the scope of the library, by incorporating new model classes.

6 Acknowledgment

I would like to thank the anonymous referees from the PANEL conferences, for their helpful remarks.

I am also very grateful to Gerardo Rubino and all the MODEL project team at the Irisa, Rennes, France, where the first draft of this paper was prepared; without their kind reception and the many discussion opportunities they generated, this work would not have been possible.

References

- [1] F. Baccelli, A. Jean-Marie, and I. Mitrani, editors. *Quantitative Methods in Parallel Systems*. Springer, 1995.
- [2] K. Bagchi and J. Walrand (eds.). *State-of-the art in performance modeling and simulation*. Gordon and Breach Books, 1996.
- [3] C. Béounes, M. Aguéra, and J. Arlat et al. SURF-2: A program for dependability evaluation of complex hardware and software systems. In *FTCS-23 Digest of Papers*, pages 668–673, Toulouse, France, June 1993. IEEE Computer Society Press.
- [4] S. Berson, E. de Souza e Silva, and R.R. Muntz. A methodology for the specification and generation of Markov models. In William J. Stewart, editor, *Numerical Solution of Markov Chains*, volume 8, chapter 2, pages 11–36. Marcel Dekker, Inc, 1991.
- [5] A. M. Blum, P. Heidelberger, S. S. Lavenberg, M. K. Nakayama, and P. Shahabuddin. System AVailability Estimator (SAVE) language reference and user's manual version 4.0. Research Report RA 219 S (82923), T. J. Watson Research Center - IBM, Yorktown Hts, NY 10598, June 1993.
- [6] M. Bouissou. The FIGARO dependability evaluation workbench in use: Case studies for fault-tolerant computer systems. In *FTCS-23 Digest of Papers*, pages 680–685, Toulouse, France, June 1993. IEEE Computer Society Press.
- [7] H. Cancela. *Dependability evaluation: combinatorial and Markovian models (Évaluation de la sûreté de fonctionnement : modèles combinatoires et markoviens)*. PhD. thesis, Rennes 1, Campus de Beaulieu, 35042 Rennes, France, 1996.
- [8] H. Cancela and G. Rubino. Model construction for system dependability evaluation (text in Spanish). In *Selected Works - VII CLAIO (Operations Research Latin-Iberian-American Congress)*, pages 235–247, Santiago, Chile, July 1994. Depto. Ingeniería Industrial, Universidad de Chile.
- [9] J. A. Carrasco and J. Figueras. METFAC: Design and implementation of a software tool for modelling and evaluation of complex fault-tolerant computing systems. In *FTCS 16 Digest of Papers*, pages 424–429, Vienna, Austria, July 1986. IEEE Computer Society Press.
- [10] G. Chiola. GreatSPN 1.5 software architecture. In *5th. Modelling Techniques and Tools for Computer Performance Evaluation*, pages 117–132, Torino, Italy, February 1991.
- [11] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, November 1993.
- [12] G. Ciardo. Discrete-time Markovian stochastic Petri nets. In *Proceedings of the 2nd. International Workshop on Numerical Solutions for Markov Chains*, pages 339–358, Raleigh, North Carolina, January 1995.
- [13] J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: applications and analysis. In *Performance'84*, pages 507–519, 1984.

- [14] W. K. Grassmann. Finding transient solutions in markovian event systems through randomization. In William J. Stewart, editor, *Numerical Solution of Markov Chains*, volume 8 of *Probability: Pure and Applied*, chapter 18, pages 37–61. Marcel Dekker, Inc, 1991.
- [15] J. Hillston. Compositional markovian modelling using a process algebra. In *Proceedings of the 2nd International Workshop on Numerical Solutions for Markov Chains*, pages 177–196, Raleigh, North Carolina, January 1995.
- [16] R.L. Klevans and W.J. Stewart. Xmarca: User’s manual version 1.0. Technical report, Dept. of Computer Science, North Carolina State University, Raleigh, North Carolina, 27695-8206, March 1991.
- [17] J.F. Meyer and W.H. Sanders. Specification and construction of performability models. In *PMCCS 2*, Mont Saint Michel, France, June 1993.
- [18] B. Plateau and K. Atif. Stochastic automata network for modeling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, October 1991.
- [19] W. H. Sanders and W.D. Obal, II. Dependability evaluation using UltraSAN. In *FTCS-23 Digest of Papers*, pages 674–679, Toulouse, France, June 1993. IEEE Computer Society Press.
- [20] W.H. Sanders and J.F. Meyer. METASAN: A performability evaluation tool based on stochastic activity networks. In *Proceedings of the Fall Joint Computer Conference*, pages 807–816, Dallas, November 1986. ACM / Computer Society of IEEE, IEEE Computer Society Press.
- [21] W. J. Stewart. MARCA: Markov Chain Analyzer, a software package for Markov modeling. In William J. Stewart, editor, *Numerical Solution of Markov Chains*, volume 8 of *Probability: Pure and Applied*, chapter 3, pages 37–61. Marcel Dekker, Inc, 1991.
- [22] W. J. Stewart, editor. *Numerical Solution of Markov Chains*, volume 8. Marcel Dekker, Inc, 1991.
- [23] J. Walrand and P. Varaiya. *High Performance Communication Networks*. Morgan Kaufmann Publishers, 1996.
- [24] M. E. Woodward. *Communication and Computer Networks - Modelling with discrete-time queues*. IEEE Computer Society Press, 1994.
- [25] A.O. Zaghoul and H.G. Perros. Approximate analysis of a shared-medium ATM switch under bursty arrivals and nonuniform destinations. *Performance Evaluation*, 21:111–129, 1994.