FACULDADE DE INFORMÁTICA

PUCRS - Brazil

http://www.pucrs.br/inf/pos/

MQNA - Markovian Queueing Networks Analyser

L. Brenner, P. Fernandes, A. Sales

Technical Report Series

Number 030
September, 2003

Contact:
lbrenner@inf.pucrs.br
www.inf.pucrs.br/~lbrenner

paulof@inf.pucrs.br
www.inf.pucrs.br/~paulof

asales@inf.pucrs.br
www.inf.pucrs.br/~asales

Abstract

This paper describes the MQNA - *Markovian Queueing Networks Analyser*, a software tool to model and obtain the stationary solution of a large class of Queueing Networks. MQNA can directly solve open and closed product-form queueing networks using classical algorithms. For finite capacity queueing models, MQNA generates Markovian description in the Stochastic Automata Networks (SAN) and Stochastic Petri Nets (SPN) formalisms. Such descriptions can be exported to the PEPS - *Performance Evaluation of Parallel Systems* and SMART - *Stochastic Model checking Analyzer for Reliability and Timing* software tools that can solve SAN and SPN models respectively. In this paper, the MQNA principles and scientific foundations are presented with examples of use, accuracy analysis, and general software characteristics.

1 Introduction

Queueing Networks (QN) are certainly the best known and most widely used formalism in performance evaluation through analytical and numerical solution. The classical product-form solutions [2] and the simplicity of the concept of queue and customers has guaranteed the popularity of Queueing Networks over the past decades. However, most "real world" problems do not fit well into the restrictions imposed by product-form assumptions, such as finite capacity.

The direct use of Markov Chains (MC) [28, 5] is a natural alternative, but the state-space explosion and the absence of product-form solutions in general prevent one from using it for realistic and large systems. Other formalisms, such as Stochastic Automata Networks (SAN) [1, 14] and Stochastic Petri Nets (SPN) [19, 21], provide some compromise between MC and QN. SAN and SPN models are simpler to describe than MC models, but their application scope is clearly much wider than the QN models. Unfortunately, SAN and SPN formalisms are not as popular as QN, and many users do not even know when a given QN model has a product-form solution or not.

In this paper, the MQNA - *Markovian Queueing Networks Analyser* software tool principles and scientific foundations are presented. MQNA is a software tool to model and obtain the stationary solution of a large class of Queueing Networks. MQNA can directly solve open and closed product-form queueing networks using classical algorithms [2, 18, 25, 20]. For finite capacity queueing models, MQNA generates Markovian description in the Stochastic Automata Networks (SAN) and Stochastic Petri Nets (SPN) formalisms. Such descriptions can be exported to the PEPS - *Performance Evaluation of Parallel Systems* [3] and SMART - *Stochastic Model checking Analyzer for Reliability and Timing* [7] software tools that can solve SAN and SPN models respectively.

The main advantage of MQNA is to provide in a single software tool the modeling of a wide class of QN models. Once defined, the QN model can be directly solved if there is a classical product-form solution. Otherwise, if the QN model has only finite capacity queues, an equivalent model can be automatically generated fot the SAN or SPN solvers (PEPS and SMART). If there is no product-form solution, nor only finite capacity queues, MQNA can generate an approximated model (with finite capacity). Such an approximation is experimental and some accuracy analysis of the technique used is presented in Section 3.4. Ultimately, it is the goal of the MQNA software tool to provide some numerical result for any QN model modeled.

The next section presents brief descriptions of the QN, SAN, and SPN formalisms. Section 3 describes the principles and scientific foundations of the MQNA software tool. Section 4 describes QN modeling examples, their conversion to SAN and SPN, and some accuracy analysis of product-form solution compared to MQNA finite capacity approximation solved in PEPS and SMART. Finally, the conclusion draws some verified advantages of the MQNA use, the foreseen application of the software, and expected evolution of new versions of MQNA.

2 Modeling Formalisms

This section briefly describes the QN formalism, the input formalism of the MQNA software tool. The formalisms SAN and SPN are presented in order to understand the translation process of QN models into Markovian representations performed by the MQNA software tool. For all three formalism the same simple example will be presented to illustrate their use.

2.1 Queueing Networks

The Queueing Networks formalism was introduced by Jackson in the 50's [17] and the major breakthroughs in this subject have been made with the product-form solutions proposed in the late 70's [2, 18, 25]. The popularity of this formalism is based on a very intuitive idea of *customers* (or *requests*) passing by *queues* (or *service centers*). A myriad of extensions to the basic formalism gives approximations [8, 9, 31] and even propose some product-form solutions [15, 12, 26, 30]. However, it is very hard to combine all of the techniques proposed in the literature to extend systematically the scope of traditional product-form queueing networks.

2.1.1 QN **Formalism**

The structure of a Queueing Network (QN) model comprises a set $\mathcal{M}$ of *queues*; a set $\mathcal{R}$ of different *classes* of customers; a function P defining the *routing probabilities* among queues, and a set $\mathcal{N}$ of the number of *customers* for each class. Thus, the structure of a Queueing Network is a four-tuple $\mathcal{QN} = (\mathcal{M}, \mathcal{R}, \mathcal{P}, \mathcal{N})$, where:

- $\mathcal{M}$ is a set of queues $\mathcal{Q}$, where $\mid \mathcal{M} \mid$ represents the number of queues in the model;
- $\mathcal{R}$ is a set of different classes of customers;
- $\mathcal{P} \subseteq \mathcal{M} \times \mathcal{M}$ with $dom(\mathcal{M})$ and $codom([0..1])$;
- $\mathcal{N}$ is a set of the number of customers for each class, where $\mid \mathcal{N} \mid = \mid \mathcal{R} \mid$.

We denote by $P^r_{i,j}$ the probability that a customer of class r, which has received service in queue Q_i, will be routed from queue Q_i to queue Q_j. These parameters can be used to compute the average visit rate of customers of class r in queue Q_i, usually denoted by V^r_i. The number of customers for each class r is denoted by N^r. For closed QN models, the values of N^r are always finite, whereas for open models these values can be infinite.

The structure of a queue comprises a maximum number K of *customers* (queue capacity); a number C of *servers*; a set $\mathcal{S}$ with *average service time* to each class of customers; a set $\mathcal{L}$ with the *average arrival rate* of customers of each class; a set $\mathcal{B}$ of behavior of customers of each class; and a set $\mathcal{D}$ of priorities among classes. Therefore the structure of a queue is a six-tuple $\mathcal{Q} = (K, C, \mathcal{S}, \mathcal{L}, \mathcal{B}, \mathcal{D})$, where:

- K is the queue capacity;
- C is the number of servers available;
- $\mathcal{S}$ is a set of average service time;
- $\mathcal{L}$ is a set of average arrival rate of customers;
- $\mathcal{B}$ is a set of behavior of customers;
- $\mathcal{D}$ is a set of discipline of service (priorities among classes of customers), where $\mid \mathcal{D} \mid = \mid \mathcal{R} \mid$.

We define K_i as the number of customers (capacity) and C_i as the number of servers available in queue Q_i. S^r_i is the average service time needed to serve one customer of class r in queue Q_i. The average arrival rate of customers of class r at queue Q_i from outside the model is denoted by L^r_i.

$B^r_{i,j}$ denotes the behavior of customers of class r routed from queue Q_i to queue Q_j when queue Q_j is full: the behavior can be *loss*[1] (the customer leaves the model), or *blocking* (queue Q_i stops the exit of customers toward queue Q_j until this queue is not full).

D_i is denoted as a set of priorities, where the value of each priority is $[1, \mid \mathcal{R} \mid]$ (the value 1 is the highest priority, whereas $\mid \mathcal{R} \mid$ is the lowest priority).

The numerical performance indices that can be computed for a QN model in MQNA are:

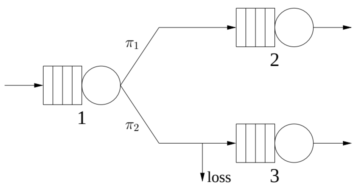
- d^r_i - average throughput of customers of class r out of queue Q_i;
- u^r_i - average utilization of servers of Q_i by customers of class r;
- n^r_i - average number of customers of class r in queue Q_i;
- w^r_i - average response time of customers of class r in queue Q_i.

The literature on queueing networks usually expresses these performance indices as functions of the model workload, *e.g.*, $d^r_i(\mathcal{N})$ for closed networks or $d^r_i(\mathcal{L})$ for open networks. However, in this paper such a distinction is not necessary and we will not indicate the model workload when mentioning the performance indices.

[1]Only open networks may have the *loss* option, since closed networks must be conservative.

2.1.2 An Example

Figure 1 is an example of an Open Queueing Network with three queues and only one single class of customers[2].



$$\begin{array}{rllll}
\mid \mathcal{M} \mid = 3 & K_1 = 3 & C_1 = 1 & S_1 = \frac{1}{\tau_1} \\
R = 1 & K_2 = 3 & C_2 = 2 & S_2 = \frac{1}{\tau_2} \\
 & K_3 = 2 & C_3 = 1 & S_3 = \frac{1}{\tau_3} \\
 & L_1 = \lambda_1 & P_{1,2} = \pi_1 & B_{1,2} = blocking \\
 & & P_{1,3} = \pi_2 & B_{1,3} = loss
\end{array}$$

Figure 1: Open Queueing Network with Loss and Blocking

The model presented in Figure 1 has one queue with a *blocking* behavior (for customers from queue 1 to queue 2) and another queue with a *loss* behavior (for customers from queue 1 to queue 3).

2.2 Stochastic Automata Networks

The Stochastic Automata Networks formalism (SAN) was proposed by Plateau [24]. The basic idea of SAN is to represent a whole system by a collection of subsystems with an independent behavior (*local transitions*) and occasional interdependencies (*functional rates* and *synchronizing events*). The SAN formalism describes a complete system as a collection of subsystems that interact with each other. Each subsystem is described as a stochastic automaton, *i.e.*, an automaton in which the transitions are labeled with probabilistic and timing information. Hence, one can build a continuous-time stochastic process[3] related to the SAN. The *global state* is the state of a SAN model defined by the combination of the *local states* of all automata. The reader interested in a formal description of the formalism can consult previous publications [14, 1].

2.2.1 SAN **Formalism**

The structure of a SAN model is a three-tuple $\mathcal{SAN} = (\mathcal{A}, \mathcal{E}, \mathcal{F})$, where:

- $\mathcal{A}$ is a set of automata;
- $\mathcal{E}$ is a set of events;
- $\mathcal{F}$ is a reachability function.

A SAN model is composed by N automata named $A^{(i)}$, with $i = 1..N$. The structure of an automaton comprises a set $\mathcal{S}$ of *states* and a transition function $\mathcal{Q}$. Therefore the structure of an automaton is a two-tuple $\mathcal{A} = (\mathcal{S}, \mathcal{Q})$, where:

- $\mathcal{S}$ is a set of states, where $\mid \mathcal{S} \mid$ represents the number of local states of the automaton;
- $\mathcal{Q}$ is a transition function, where $\mathcal{Q} \subseteq \mathcal{S} \times \mathcal{S}$ with $dom(\mathcal{S})$ and $codom(\mathcal{T})$, and $\mathcal{T}$ is a set of event label with probabilities[4], *i.e.*, (e, π_e), where $e \in \mathcal{E}$, and $\pi_e \in [0..1]$.

Set $\mathcal{E}$ of events is composed of E events labeled e_j, with $j = 1..E$. Each event e_j is defined by an identifier e and a firing rate τ_e. Firing rate τ_e is defined as a functional element. A functional element $f(\mathcal{A}^{(\phi)})$ is a function of $\prod_{i \in \phi} S^{(i)}$ in $\mathbb{R}^+$, where ϕ is a subset of $[1..N]$. Automata $\mathcal{A}^{(i)}$ with $i \in \phi$ are the parameters of the element $f(\mathcal{A}^{(\phi)})$ that is evaluated relative to the local states $x^{(\phi)}$. Functional rates or probabilities are utilized to represent interaction among automata.

There are two types of events that change the global state of a SAN model: *local events* and *synchronizing events*. Local events change the global state passing from a global state to another that differs only by one local state. On the other hand, synchronizing events can change simultaneously

[2]In single-class models, class index r will be omitted for the service times (S_i^r), the routing probabilities (P_{ij}^r), the arrival rates (L_i^r), and the behaviors of customers (B_{ij}^r).

[3]Queueing Networks formalism usually describes the inter-arrival and service duration with a continuous-time scale. Therefore, in the context of this paper only continuous-time SAN will be considered, although discrete-time SAN can also be employed without loss of generality.

[4]A same event can lead to more than one local state. Consequently, a "choice" probability must be specified to each event occurrence.

more than one local state, *i.e.*, two or more automata can change their local states simultaneously. In other words, the occurrence of a synchronizing event *forces* all automata concerned to fire a transition corresponding to this event. Thus, local events can be viewed as a particular case of synchronizing events that concerns only one automaton.

The combination of local states of each automaton ($\prod_{i=1}^{N} S^{(i)}$) defines the state space with all global states, which is called the *product state space*. However, some global states may not represent valid model situations, defining a reduction in the product state space. This reduced state space is composed only by reachable global states, and, hence, called *reachable state space*.

Reachability function $\mathcal{F}$ defines the reachable state space by associating a value 1 to each reachable global state of $\prod_{i=1}^{N} S^{(i)}$, and a value 0 otherwise.

2.2.2 An Example

Figure 2 presents the SAN model equivalent to the QN model in Figure 1. Three automata ($A^{(1)}$, $A^{(2)}$ and $A^{(3)}$), three local events, and two synchronizing events were used for this example. Synchronizing events e_{12} and e_{13} respectively represent the routing of customers from Q_1 to Q_2 and from Q_1 to Q_3. The arrival of customers in Q_1, and the departure of customers from Q_2 and Q_3 are represented by local events l_1, m_2 and m_3 respectively. Table 1 presents the rates of all events of this example.

Local		Synchronizing	
event	rate	event	rate
l_1	λ_1	e_{12}	$\tau_1\pi_1$
m_2	$min(C_2, st(\mathcal{A}^{(2)})\tau_2$	e_{13}	$\tau_1\pi_2$
m_3	τ_3		

Table 1: Local and synchronizing events of Figure 2

Local event m_2 has rate equal to τ_2 when automaton $A^{(2)}$ is in local state $1^{(2)}$ and rate equal to $2\tau_2$ when it is in local states $2^{(2)}$ and $3^{(2)}$. To correctly describe that, event m_2 has a functional rate. The other events have constant rates.

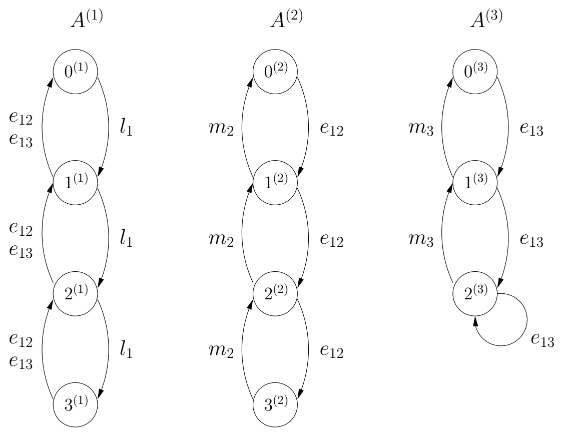


Figure 2: SAN model for an Open Queueing Network with Loss and Blocking

The occurrence of event e_{12} forces the simultaneous change of automata $A^{(1)}$ and $A^{(2)}$. Synchronizing event e_{12} cannot occur when automaton $A^{(2)}$ is in local state $3^{(2)}$ (Q_2 is full). Hence, it corresponds to the *blocking* of the departure of customers from Q_1 to Q_2. An extra loop transition in the last state of automaton $A^{(3)}$ (event e_{13}) allows the departure of customers from Q_1 without an arrival in Q_3, *i.e.*, the *loss* of customers in Q_3 (Q_3 is full).

The performance indices of this model are extracted from the marginal probability of each queue. The integration function to compute the average population of Q_i is: "$st(A^{(i)})$". The other performance indices follow the same idea, *e.g.*, the throughput of each Q_i is calculated with the function $\frac{min(C_i, st(A^{(i)}))}{S_i}$.

2.3 Stochastic Petri Nets

The Stochastic Petri Nets (SPN) formalism has been defined independently by several authors [27, 29, 4, 21] in the late seventies. The basis of this formalism is the well-known Petri nets, which is based on the automata representation using a graph with two types of nodes (*places* and *transitions*) [23, 22]. The major addition of the SPN formalism to the standard Petri nets is the assignment of an exponentially distributed random firing time to each transition. SPN formalism is a powerful tool for modeling and evaluating the performance of systems involving concurrency, nondeterminism, and synchronization. The major drawback of SPN is the large state space that can be generated even by rather simple models.

2.3.1 SPN **Formalism**

The structure of SPN formalism comprises a set of *places* $\mathcal{P}$; a set of *transitions* $\mathcal{T}$; three functions of transitions into a set of places ($\mathcal{P}^*$): I, O, and H, respectively defining the input, output and inhibition places of a given transition; a time function W associating a positive and non-zero rate to each transition; G is a function that assigns a list of firing conditions to each transition; and initial marking M_0 of each place. Formally, a SPN model is an eight-tuple $\mathcal{SPN} = (\mathcal{P}, \mathcal{T}, I, O, H, W, G, M_0)$, where:

- $\mathcal{P}$, a set of places;
- $\mathcal{T}$, a set of transitions[5];
- I, O, and H: $\mathcal{T} \rightarrow \mathcal{P}^*$, the input, output, and inhibition functions of the transitions into a set of places;
- W: $\mathcal{T} \rightarrow \mathbb{R}^{*+}$, the function associating an exponentially distributed stochastic process to each transition[6];
- G: the function associating a list of conditions, called *guards*, to each transition;
- M_0: $\mathcal{P} \rightarrow \mathbb{N}$, the initial number of tokens in each place.

2.3.2 An Example

Figure 3 presents the SPN model equivalent to the QN model in Figure 1. Each queue i is equivalent to a pair of places Q_i and A_i. Place A_i represents the available place for customers in queue i (capacity), and place Q_i represents the current customers in queue i.

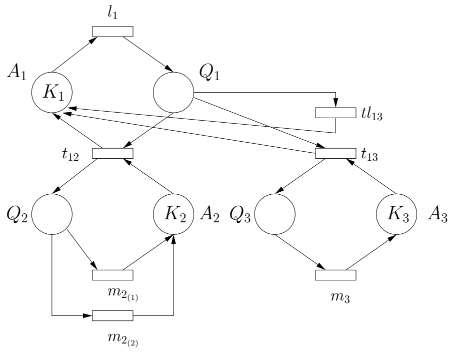


Figure 3: SPN model for an Open Queueing Network with Loss and Blocking

Transition l_1 has rate equal to λ_1 and it represents the arrival in Q_1. Transition $m_{2_{(1)}}$ has rate equal to τ_2 and transition $m_{2_{(2)}}$ has rate equal to $2\tau_2$. Transition $m_{2_{(1)}}$ symbolizes the departure of customers from Q_2 when there is only one token in it, whereas transition $m_{2_{(2)}}$ symbolizes the departure of customers from Q_2 when there is two or more tokens in it. A *guard* (see Section 3.3.3) is specified to express a choice between these two transitions. The *guard* with condition equal to $tk(Q_2) == 1$ is specified to transition $m_{2_{(1)}}$, whereas a *guard* with condition equal to $tk(Q_2) > 1$ is

[5]Obviously, sets $\mathcal{P}$ and $\mathcal{T}$ must not be empty, otherwise the SPN is meaningless.
[6]The numeric value expressed is the average rate of occurrence of the transition.

specified to transition $m_{2_{(2)}}$. Transition m_3 has rate equal to τ_3 and represents the departure from Q_3.

Synchronizing transition t_{12} has rate equal to $\tau_1\pi_1$ and represents the routing of customers from Q_1 to Q_2. Analogously, t_{13} represents the routing of customers from Q_1 to Q_3 and its rate is equal to $\tau_1\pi_2$. Transition tl_{13} symbolizes the loss of customers from Q_1 and its rate is also equal to $\tau_1\pi_2$, but this transition can only be fired if the third queue is full. Thus, a *guard* with condition $tk(A_3) == 0$ (the queue is full, since there is no tokens in the *available* place A_3) must be assigned to transition tl_{13}.

3 MQNA Software Tool

The objective of the MQNA [6] is to provide in a single software environment the modeling and stationary solution of a rather generical class of Queueing Networks (QN) models. The class of QN models handled by MQNA includes some classical product-form models [18, 2, 25] and some finite capacity models [13]. For these classes of QN models, respectively called PFQN and FCQN, the MQNA software computes the exact stationary solution (for PFQN models) or generates exact Markovian representation (for FCQN models) using SAN and SPN formalisms. Some QN models handled by MQNA do not belong to either of these two classes. For such models, MQNA provides an approximate (within a tolerance) finite capacity queue representation.

The next four sections describe briefly:

- the product-form solutions used by MQNA to solve PFQN models;
- the conversion of FCQN models into a SAN model;
- the conversion of FCQN models into a SPN model;
- the approximation performed by MQNA to transform other models (neither FCQN, nor PFQN models) into FCQN models.

3.1 Classical Product-Form Solution used in MQNA

The product-form solutions in MQNA do not cover all known solutions in literature, *e.g.* [15, 12, 26, 30]. Only the classical solutions first expressed by Baskett, Chandy, Muntz and Palacios [2] are implemented. The algorithms of solutions employed are based on the arrival theorem, and Little's law expressed according to the original definition for open QN [2] and in the Mean Value Analysis for closed QN [25].

For open PFQN models, MQNA computes exact stationary solution for multi-class, mono-server (load independent) and infinite capacity queues. Such restriction requires a limited workload, *i.e.*, an arrival rate of customers inferior to the service rate of each queue. Due to the same restriction, such models are free from *blocking* or *loss* behaviors, since the queues always accept new customers.

For closed PFQN models, MQNA computes exact stationary solution for multi-class, multi-server (load dependant) and *virtually infinite* capacity queues, *i.e.*, the queues capacities are always enough to receive all customers in the model[7]. It is important to notice that closed PFQN models can be exactly represented by a FCQN model, and, therefore, can be translated to a finite Markovian representation in SAN or SPN formalism. On the other hand, open PFQN models can only be approximated by a finite Markovian representation as will be seen in Section 3.4.

3.2 Conversion of FCQN to SAN

The main idea of the conversion method is to create for each queue i and class r a stochastic automaton in the SAN model. This method is particularly interesting for QN that has no product-form solution, *e.g.*, open models with limited capacity queues. However, closed product-form queueing networks can be converted to a SAN model exactly according to the PFQN specification, since the queue's capacities are equal to the total number of customers in the network. The required steps to convert a QN model to SAN model are described below.

[7]Closed queueing models are conservative and, therefore, have a finite number of customers in the network.

3.2.1 Queue Modeling

One automaton models each queue i with only class r customers. Each automaton has $K + 1$ states, where K is the queue's capacity. Each state $x^{(i^r)}$ represents the possible number of customers in the queue.

3.2.2 Arrival and Departure of Customers to/from the Model

The arrival of customers from the exterior of class r in queue i is modeled by a local event with rate L_i^r associated with a transition that makes the automaton change from state $x^{(i^r)}$ to state $x^{(i^r)} + 1$. On the other hand, the departure of customers is represented by a local event that changes the automaton state from $x^{(i^r)}$ to $x^{(i^r)} - 1$. The rate of this event is modeled by the function:

$$\left(\frac{min(C_i, st(\mathcal{A}^{(i^r)})}{S_i^r}\right) \times P_{ij}^r \times h_i^r$$

This function takes into account the number of customers in queue i $(st(\mathcal{A}^{(i^r)}))$ and the servers (C_i) of queue i; the service time (S_i^r); the routing probability (P_{ij}^r); and the service priority function (h_i^r) (described in Section 3.2.4).

3.2.3 Routing

The routing of class r customers from queue i to queue j is made by a synchronizing event. This synchronizing event concerns only two automata: automaton $\mathcal{A}^{(i^r)}$ and $\mathcal{A}^{(j^r)}$. The rate of such an event is given by the function:

$$\left(\frac{min(C_i, st(\mathcal{A}^{(i^r)})}{S_i^r}\right) \times P_{ij}^r \times h_i^r \times f_i^r$$

The first part of this function is exactly identical to the previous one (for local events of arrival and departure from/to exterior of the model). The only difference is the inclusion of function f_i^r to represent the (possible) blocking behavior due to the limited capacity of queue j. This function is detailed in the next section.

3.2.4 Restrictions

Queue Capacity. When representing a multi-class QN model, each automaton represents the total queue capacity. Thus, the real capacity of the queue is modeled $\mid \mathcal{R} \mid$ times, where $\mid \mathcal{R} \mid$ is the number of classes in the model. To correct this problem, the arrival rates (local events) and service rates (synchronizing events) are multiplied by a *boolean function*. This boolean function evaluates whether the target queue is full or not. If the queue is not full, the transition fires normally at the expected rate. Otherwise, if the queue is full, there are two different behaviors: *blocking* or *loss*.

The *blocking* behavior inhibits the event occurrence if the target queue (queue j) is full. In this case function f_i^r must return a zero value to inhibit the service in queue i, or the value 1 to allow the service, *i.e.*:

$$f_i^r = \left(\sum_{r=1}^{|\mathcal{R}|} st(\mathcal{A}^{(i^r)})\right) < K_i$$

When a routing has a *loss* behavior, the transitions must be fired in both concerned automata, but the number of customers only changes (decreases) in the automaton representing queue i. In order to do that, *loop transitions* must be included in all states[8] of automaton $\mathcal{A}^{(j^r)}$.

Service Priority. Although a myriad of service priority behaviors could be modeled in SAN, the MQNA implements only two kinds of service priorities behaviors:

- a class has absolute service priority over another class, *i.e.*, customers of the non-priority class cannot be served unless there are no customers of the priority class in the queue; and
- the customers of the classes do not have service priority among themselves, *i.e.*, the actual average service rate of the queue (which is different for customers of each class) will be proportional to the number of customers of each class present in the queue.

[8]If queue j is visited by only one class, only the last state must have a loop transition.

For the first case, if class r_1 customers have priority over class r_2 customers, function $h_i^{r_2}$ must be:

$$h_i^{r_2} = st(\mathcal{A}^{(i^{r_1})}) == 0$$

For the second case, considering both classes with equal priority, the (non) priority functions must be:

$$h_i^{r_1} = \frac{st(\mathcal{A}^{(i^{r_1})})}{st(\mathcal{A}^{(i^{r_1})} + st(\mathcal{A}^{(i^{r_2})})} \qquad h_i^{r_2} = \frac{st(\mathcal{A}^{(i^{r_2})})}{st(\mathcal{A}^{(i^{r_1})} + st(\mathcal{A}^{(i^{r_2})})}$$

3.3 Conversion of FCQN to SPN

The basic principle of this conversion method is to create a set of places (which represent the queue states), a set of arrival transitions, and a set of departure transitions. The set of arrival transitions represents the arrival of customers in the queue (arrival of customers from the exterior or routing from another queue). Analogously, the set of departure transitions symbolizes the departure of customers from the queue (departures of customers to the exterior or routing to another queue). We will describe in the next sections the required steps to convert a QN model to a SPN model.

3.3.1 Queue Modeling

A set of places models each queue i of class r. This set of places has one place A_i that symbolizes the available capacity in queue i (considering the number of customers already in it), and $\mid \mathcal{R} \mid$ (number of classes) places Q_i^r that expresses the number of customers of class r in queue i. This set of places (A_i and all Q_i^r) is a *P-invariant* with a token count equal to K_i [22].

3.3.2 Arrival and Departure of Customers to/from the Queue

In the same way as for the conversion to SAN, the arrival of class r customers from the exterior in queue i is modeled by a transition denoted as l_i^r. This transition removes tokens from place A_i and generates new ones in place Q_i^r. Obviously, transition l_i^r belongs to the set of arrival transitions of queue i and its rate is L_i^r.

Similarly, the departure of class r customers in queue i is modeled by a transition denoted as m_i^r (when it represents the departure to the exterior) or t_{ij}^r (when it represents routing from queue i to queue j). These transitions belong to the set of departure transitions of queue i and they remove tokens from place Q_i^r and generate new ones in place A_i.

The multi-server (load dependent) queues require an additional complexity, since there must be as many departure transitions as the number of servers C_i of the queue. The rate of each of these transitions is in general:

$$\left(\frac{min(C_i, tk(Q_i^r))}{S_i^r}\right) \times P_{ij}^r,$$

where $tk(Q_i^r)$ represents the number of the tokens (customers) in place Q_i^r. Every one of these departure transitions must have a *guard* specifying which of them can happen at each queue state (number of customers present in the queue).

It is important to notice that, unlike to the conversion to SAN, in which the service rate is functional, the SPN model describes the departure with alternative, but constant rate transitions.

3.3.3 Restrictions

Queue Capacity. The queue capacity of multi-class QN model using SPN formalism is represented by the number of tokens in place A_i. As mention before, a queue may have two different behaviors: *blocking* or *loss*. *Blocking* is the default behavior of the queue and there is no restriction to represent it in the model, since the number of tokens in place A_i is the available capacity of customers in the queue. It is necessary to create another departure transition tl_{ij}^r to represent the *loss* behavior, when queue j is full. This transition removes tokens from place Q_i^r and generates new ones in place A_i, but there is a *guard* specifing to transition tl_{ij}^r with a condition $tk(A_j) == 0$, *i.e.*, transition tl_{ij}^r only can fire if queue j is full.

Service Priority. Function *guard* may be used to define a dynamic behavior of the model. It specifies a necessary given condition to fire a given transition. The priority of a class, or even the absence of priority, may be modeled in SPN using function *guard*. When, in a queue i, class r_1 customers have service priority over class r_2 customers, it is necessary to assign a *guard* to transition $t_{ij}^{r_2}$ with condition $tk(Q_i^{r_1}) == 0$, *i.e.*, there is no customers of class r_1 in queue i. The absence of priority is a little bit more complex, since it becomes necessary to create a great number of departure transitions, each with a different average service rate and corresponding guard, to each possible combination of number of customers of each class. In fact, this practice is equivalent to create a departure transition to each possible evaluation of the functional rates used in the conversion to SAN model presented in Section 3.2.4.

3.4 Approximation to Finite Capacity Queues

The conversions described in the previous sections can be performed exactly only when all queues have a finite, and preferably small, capacity. However many queueing networks modeled by MQNA do not respect this restriction, neither have classical product-form solution[9]. Such models cannot be converted to an exactly equivalent finite Markovian description (in SAN or in SPN). Nevertheless, the software tools to solve finite Markovian description (PEPS or SMART) use iterative methods to find approximated solutions within a given tolerance. Therefore, any approximation limiting the capacity of some queues may have little impact in the precision of the solution.

The basic idea to perform this approximation is to limit, to each queue, the capacity to a minimum value. In such way that the number of local states of a queue will be reduced. A good approximation must guarantee that the sum of the eliminated states probability will be inferior to the tolerance accepted in the subsequent iterative method to found the stationary solution (in PEPS or in SMART).

3.4.1 Single-Class Models

Initially, the single-class models will be considered. Individually, any single-server ($C_i = 1$) queue i can be viewed as a load independent birth-and-death process [5], in which the arrival (λ_i) and service rate (μ_i) can be approached by the queue throughput (d_i) and the inverse of service time ($1/S_i$), respectively. Therefore, the accumulated probability of the queue states until the $k+1$-th state (from 0 customers until k customers in the queue) will be given by:

$$\left(1 - \frac{\lambda_i}{\mu_i}\right) + \sum_{j=1}^{k} \left(\frac{\lambda_i}{\mu_i}\right)^j$$

When the difference of such value and 1 is less than the accepted tolerance, the queue capacity reduction to k must not represent an important loss of accuracy in the stationary solution obtained by the software PEPS or SMART using the same tolerance.

Considering now the case of multi-server queues, the same approach can be used, but with underestimated result, *i.e.*, obtaining a queue capacity reduction smaller than the possible one. Multi-servers queues, compared to single-servers queues, will have an average service rate higher than the inverse of service time ($1/S_i$). Therefore the actual accumulated probability of the queue states until the $k+1$-th state will be overestimated by the previous equation.

3.4.2 Multi-Class Models

The multi-class models with no priority can be approximated similarly adapting the estimation of the arrival (λ_i) and service (μ_i) rates of each queue i respectively by the sum of throughput of all classes, and a proportion of all the service rates by the throughput of each class, *i.e.*:

$$\lambda_i = \sum_{\forall r} d_i^r \qquad \mu_i = \frac{\sum_{\forall r} \left(d_i^r \frac{1}{S_i^r}\right)}{\sum_{\forall r} d_i^r}$$

The multi-class models with priority among the classes can be approached using the same method, if we consider queue i not congested, *i.e.*, with $\lambda_i < \mu_i$. In this low congestion situation, it is reasonable to admit that all customers will be served. If the queue is rather congested ($\lambda_i \geq \mu_i$), most likely the

[9]This is the case of any open model with some queues with finite capacity, and some others with infinite capacity.

queue i cannot have its capacity reduced. In this high congestion situation, the method to estimate a capacity reduction becomes irrelevant due to the higher probability of the states with a greater number of customers. Therefore, using the same formula for multi-class models with or without priority must not induce errors in the approximation technique.

The numerical achievements of such technique can be observed in detail in the accuracy comparison in Section 4.2. It is important, nevertheless, to keep in mind that such approximation is experimental, and its goal is to provide some numerical results for hopeless case models, *i.e.*, models that cannot be solved by product-form algorithms, nor by exactly equivalent conversion to SAN or SPN.

4 Modeling Examples

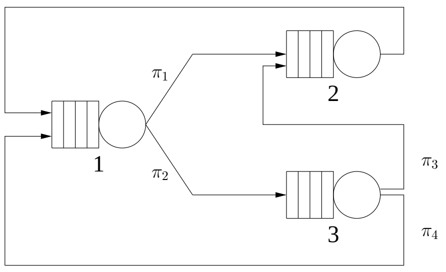
The four examples presented in this section intend to show the features available in the MQNA software. All examples in this section, and many others, are available in the MQNA software page [6].

4.1 Conversion of Equivalent FCQN Models

The next two examples intend to show the equivalence of some QN models to the equivalent SAN and SPN models obtained by MQNA. It is not our intention to show the capabilities to handle large models, but to precisely describe two conversion cases. The examples in the next section have a more important size in order to analyse the MQNA, and also PEPS and SMART, capabilities to handle large models.

4.1.1 Closed Product-Form Queueing Network

This example (Figure 4) is a simple Closed Queueing Network model with product-form solution.



$$\begin{array}{rlll} \mid \mathcal{M} \mid = 3 & C_1 = 1 & S_1 = \frac{1}{\tau_1} \\ R = 1 & C_2 = 2 & S_2 = \frac{1}{\tau_2} \\ N = 4 & C_3 = 1 & S_3 = \frac{1}{\tau_3} \\ & P_{1,2} = \pi_1 & P_{3,2} = \pi_3 \\ & P_{1,3} = \pi_2 & P_{3,1} = \pi_4 \end{array}$$

Figure 4: Closed Product-Form Queueing Network

The proposed SAN model (Figure 5) has one automaton to each queue and five synchronizing events (e_{12}, e_{13}, e_{21}, e_{31} and e_{32}) representing the five routing paths of the model.

Each automaton represents a queue with capacity to hold up to four customers, *i.e.*, no blocking behavior will be necessary, since the network has only four customers ($N = 4$). This model has no local events. The departure of a customer from a queue always represents the arrival in another queue.

Event e_{12} has rate equal to $\tau_1\pi_1$ and event e_{13} has rate equal to $\tau_1\pi_2$. Event rate e_{21} is equal to τ_2 when automaton $A^{(2)}$ is in local state $1^{(2)}$ and it is equal to $2\tau_2$ when it is in local states $2^{(2)}$, $3^{(2)}$ and $4^{(2)}$. Event rate e_{31} is equal to $\tau_3\pi_4$, as well as event e_{32} has rate equal to $\tau_3\pi_3$.

The product state space of this model is considerably larger than the reachable state space, since it models in the product state space all possible combinations of the local states. However, only the combinations of the local states in which the sum corresponds to the total number of customers N are reachable. For this example (with $N = 4$), the product state space results in 125 states, but only 15 are reachable. The reachability function of the model of Figure 5 is:

$$reachability = \left(\sum_{i=1}^{|\mathcal{M}|} st(\mathcal{A}^{(i)}) \right) == N$$

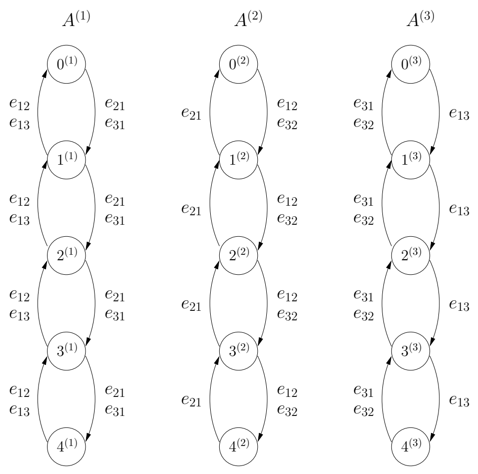


Figure 5: SAN model for a Closed Product-Form Queueing Network

The suggested SPN model (Figure 6) has one place A_i, places Q_i to each queue i, and six synchronizing transitions (t_{12}, t_{13}, $t_{21_{(1)}}$, $t_{21_{(2)}}$, t_{31} and t_{32}). Places Q_i and A_i are a *P-invariant* with a token count equal to K_i. M_i represents the number of tokens in the place Q_i (customers in queue). It may be a number between zero and K_i, but the sum of M_i must be equal to N (total number of customers).

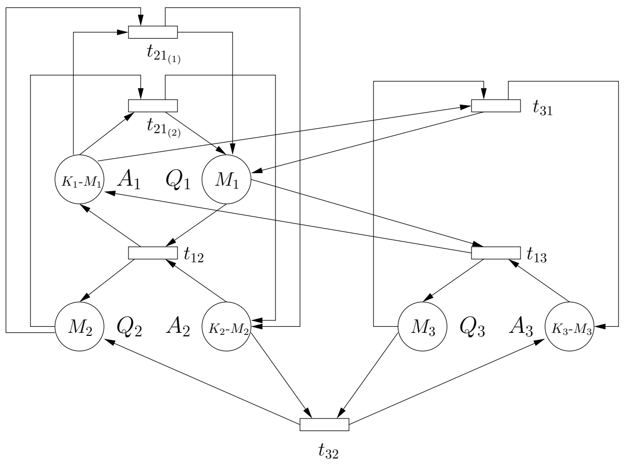


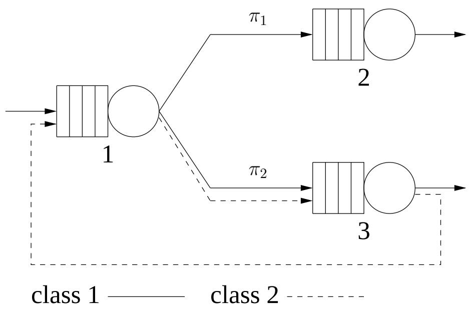
Figure 6: SPN model for a Closed Product-Form Queueing Network

All transitions of this example are synchronizing transitions, since the departure of a customer from a queue always represents the arrival in another queue. Transition t_{12} has rate equal to $\tau_1\pi_1$ and it symbolizes the passage of customers from Q_1 to Q_2, as well as t_{13} is the passage of customers from Q_1 to Q_3 and has rate equal to $\tau_1\pi_2$. Transition $t_{21_{(1)}}$ has rate equal to τ_2 and transition rate $t_{21_{(2)}}$ is equal to $2\tau_2$. Transition $t_{21_{(1)}}$ represents the passage of customers from Q_2 to Q_1 when there is only one token in it, whereas transition $t_{21_{(2)}}$ represents the passage of customers from Q_2 to Q_1 when there is two or more tokens in it. A *guard* (see Section 3.3.3) is specified to transition $t_{21_{(1)}}$ with condition equal to $tk(Q_2) == 1$, and another *guard* is specified to transition $t_{21_{(2)}}$ with condition equal to $tk(Q_2) > 1$. Synchronizing transition rate t_{31} is equal to $\tau_3\pi_4$ and it represents the passage of customers from Q_3 to Q_1, as well as t_{32} is the passage of customers from Q_3 to Q_2 and has rate equal to $\tau_3\pi_3$.

4.1.2 Mixed Queueing Network

The second example (Figure 7) is a mixed network in which the first class of customers routing pattern is an open system and the second class of customers routing pattern is a closed system.

The construction technique of such model does not differ significantly from the technique employed in the previous models. A local event is used to represent each communication with the exterior (arrival at Q_1^1 and departure of customers of class 1 from Q_2^1 and Q_3^1). Synchronizing events are used to represent the exchange of customers from one queue to another. Functional transitions are used to represent the capacity restriction of admission in queues accepting both classes of customers. Functional transitions are also used to express the priority among queues.



$\mid \mathcal{M} \mid = 3$	$K_1 = 3$	$C_1 = 1$	$S_1^1 = \frac{1}{\tau_1}$
$R = 2$	$K_2 = 2$	$C_2 = 1$	$S_2^1 = \frac{1}{\tau_2}$
$N^2 = 4$	$K_3 = 3$	$C_3 = 1$	$S_3^1 = \frac{1}{\tau_3}$
	$P_{1,2}^1 = \pi_1$	$D_1 = \{1,2\}$	$S_1^2 = \frac{1}{\tau_4}$
	$P_{1,3}^1 = \pi_2$	$D_3 = \{1,2\}$	$S_3^2 = \frac{1}{\tau_5}$
	$L_1^1 = \lambda_1$		

Figure 7: Mixed Queueing Network

The equivalent SAN model (Figure 8) has five automata ($A^{(1^1)}$, $A^{(2^1)}$, $A^{(3^1)}$, $A^{(1^2)}$, $A^{(3^2)}$), three local events (arrival and departures of customers of class 1), and four synchronizing events (the routing paths for customers of both classes).

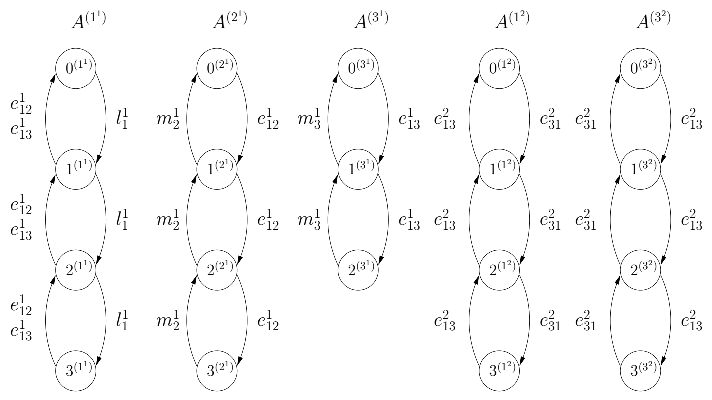


Figure 8: SAN model for a Mixed Queueing Network

Functions f_1 and f_2 are used to express the fact that a queue has reached its capacity. For this reason, we define these two functions as:

- $f_1 = (st(\mathcal{A}^{(1^1)}) + st(\mathcal{A}^{(1^2)})) < K_1$;
- $f_2 = (st(\mathcal{A}^{(3^1)}) + st(\mathcal{A}^{(3^2)})) < K_3$;

The priority of customers of class 1 over class 2 at Q_1^1 and Q_3^1 is expressed by functions h_1 and h_2 respectively. The customers of class 2 in Q_1^1 and Q_3^1 may be served only if there are no customers of class 1. Therefore, function h_1 (h_2) services the customers of class 2 only when automaton $\mathcal{A}^{(1^1)}$ ($\mathcal{A}^{(3^1)}$) is in local state $0^{(1^1)}$ ($0^{(3^1)}$). The functions are defined as:

$$h_1 \; = \; st(\mathcal{A}^{(1^1)}) == 0 \qquad\qquad h_2 \; = \; st(\mathcal{A}^{(3^1)}) == 0$$

Table 2 shows the corresponding values of all local and synchronizing events used in the SAN model (Figure 8).

Class 1				Class 2	
Local		Synchronizing		Synchronizing	
event	rate	event	rate	event	rate
l_1^1	$\lambda_1 f_1$	e_{12}^1	$\tau_1\pi_1$	e_{13}^2	$\tau_4 f_2 h_1$
m_2^1	τ_2	e_{13}^1	$\tau_1\pi_2 f_2$	e_{31}^2	$\tau_5 f_1 h_2$
m_3^1	τ_3				

Table 2: Local and synchronizing events of Figure 8

An important remark about the proposed model is that the product state space of the SAN model is greater than the reachable state space. The reachability function of the SAN model equivalent to the network of Figure 7 must take into account: the unreachable states due to the use of two automata to represent a queue accepting two classes of customers; and the conservative number of customers of class 2. The equivalent SAN model has a 768 product state space but only 24 states are reachable. The reachability function for this model is:

$$\begin{aligned} reachability \quad = \quad & \left(\left(st(A^{(1^1)}) + st(A^{(1^2)})\right) \leq 3\right) \text{ and} \\ & \left(\left(st(A^{(3^1)}) + st(A^{(3^2)})\right) \leq 3\right) \text{ and} \\ & \left(\left(st(A^{(1^2)}) + st(A^{(3^2)})\right) == 4\right) \end{aligned}$$

The proposed SPN model (Figure 9) has two classes, eight places (Q_1^1, Q_2^1, Q_3^1, Q_1^2, Q_3^2, A_1, A_2, A_3) and seven transitions (l_1^1, m_2^1, m_3^1, t_{12}^1, t_{13}^1, t_{13}^2, t_{31}^2). In this example, queues 1 and 3 are represented by places Q_1^1, A_1 and Q_1^2 (queue 1), and by places Q_3^1, A_3 and Q_3^2 (queue 3). Queue 2 is only represent by places Q_2^1, A_2, since it has one single class.

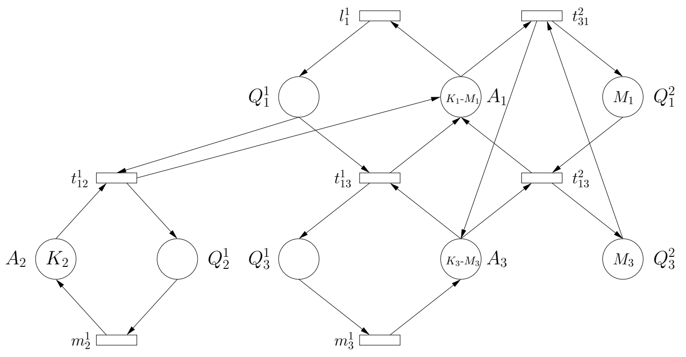


Figure 9: SPN model for a Mixed Queueing Network

Transition l_1^1 represents the arrival of customers in place Q_1^1. It has arrival rate equal to λ_1. Transition t_{12}^1 has rate equal to $\tau_1\pi_1$ and it symbolizes the passage of customers from Q_1^1 to Q_2^1. Transition t_{13}^1 is the passage of customers from Q_1^1 to Q_3^1 and has rate equal to $\tau_1\pi_2$. Transitions m_2^1 and m_3^1 symbolize the departure of customers from Q_2^1 and Q_3^1 respectively. Transition rate m_2^1 is equal to τ_2, whereas m_3^1 has rate equal to τ_3. Transitions t_{13}^2 and t_{31}^2 have rate equal to τ_4 and τ_5 respectively. These transitions represent the passage of customers from Q_1^2 to Q_3^2 and vice-versa. A *guard* is specified to transitions t_{13}^2 and t_{31}^2. A *guard* with condition equal to $tk(Q_1^1)$ == 0 to transition t_{13}^2, and another with condition equal to $tk(Q_3^1)$ == 0 to transition t_{31}^2. These *guards* represent the service priority of class 1 over class 2 in queue 1 and 3.

4.2 Conversion of Approximation to Finite Capacity Queues

The approximation technique explained in Section 3.4 considers each queue behavior individually. Obviously, the effects of blocking and loss in customers routing is not taken into account.

The purpose of this section is to verify the impact of this approximation in the computational of stationary solutions. The set of examples tested include some variations of input parameters for one PFQN model, and a quite complex example with limited, but large, capacity queues (a FCQN model). The main application of the approximation technique is for models that do not belong to these two classes (PFQN and FCQN). However, for these two examples the accuracy of the approximated models can be compared to the exact solution of the original models.

4.2.1 Single-Class Open PFQN

The first example is an open PFQN similar to the example carried out through Section 2 (Figure 1). To have a product-form solution, all queues will be considered with infinite capacity ($K_1 = K_2 = K_3 = \infty$) and with single-server ($C_1 = C_2 = C_3 = 1$). Consequently, this model will not have blocking, nor loss behavior. To this example three sets of numerical parameters will be considered, varying the service time of each queue.

- A - all queues with a very low utilization index ($u_1 \simeq 0.4$, $u_2 \simeq 0.3$, and $u_3 \simeq 0.1$):
  $S_1 = 0.2$; $S_2 = 0.25$; $S_3 = 0.2$;
- B - the first and second queue with a high utilization index, the third queue with low utilization indices ($u_1 \simeq 0.8$, $u_2 \simeq 0.4$, and $u_3 \simeq 0.1$):
  $S_1 = 0.4$; $S_2 = 0.5$; $S_3 = 0.2$;
- C - all queues with a high utilization index ($u_1 \simeq 0.8$, $u_2 \simeq 0.7$, and $u_3 \simeq 0.6$):
  $S_1 = 0.4$; $S_2 = 0.5$; $S_3 = 1.0$;

The workload of the model (arrival in the first queue - L_1) and the visit rate of the queues will be the same for all three sets of parameters.

Set of Parameters	K_1	K_2	K_3	*pss*	*precision*
exact	∞	∞	∞	∞	10 digits
A	26	22	11	7,452 states	7 digits
B	104	65	11	83,952 states	5 digits
C	104	65	46	325,710 states	4 digits

Table 3: Accuracy of approximation for a single-class open PFQN

Using a tolerance of 10^{-10} for the choice of limited capacity and for convergence in PEPS and SMART, Table 3 presents the capacity estimated for each queue (K_1, K_2, and K_3), the model product state space (*pss*), and the number of correct digits in computing the average length of each queue (*precision*)[10]. The first row of the table indicates the precision of the product-form solution. Observing this table, it is clear that the approximation works better for non-congested models. As expected, the results limiting the capacity of each queue can be estimated independently, even if the gains in precision are rather equally observed in all queues.

4.2.2 Two-Class Mixed FCQN

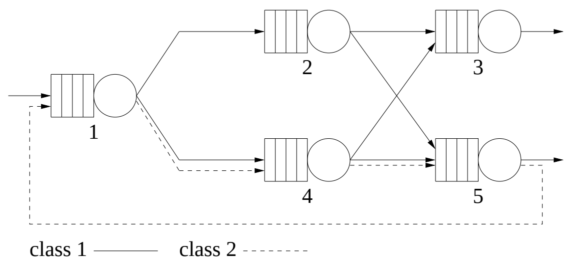
The second example will be a two-classes mixed FCQN with a rather large product state space. For this model (Figure 10), customers of the first class will act as an open system, and the customers of the second class will act as a closed system. This model is a larger version of the model of Figure 7 presented in Section 4.1.2, in which there are five queues.

For this example, also three sets of parameters will be considered.

- A - both classes with high workload demand:
  $L_1^1 = 0.8$ and $N^2 = 10$
- B - first class with low workload demand, second class with high workload demand:
  $L_1^1 = 0.4$ and $N^2 = 10$
- C - both classes with low workload demand:
  $L_1^1 = 0.4$ and $N^2 = 5$

Using the same tolerance as in the previous section, Table 4 presents the capacity used for each queue (K_1, K_2, K_3, K_4, and K_5), the product (*pss*) and the reachable (*rss*) state space of the model,

[10]The number of digits indicated is the worst case, *i.e.*, the minimum number of correct digits for computing the average queue length for the queue with the largest error.



$\mid \mathcal{M} \mid = 5$	$K_1 = 10$	$C_1 = 1$	$S_1^1 = 0.1$
$R = 2$	$K_2 = 10$	$C_2 = 1$	$S_2^1 = 0.1$
	$K_3 = 10$	$C_3 = 1$	$S_3^1 = 0.2$
	$K_4 = 8$	$C_4 = 1$	$S_4^1 = 0.05$
	$K_5 = 8$	$C_5 = 1$	$S_5^1 = 0.15$
	$P_{1,2}^1 = 0.7$	$P_{2,5}^1 = 0.5$	$S_1^2 = 8.0$
	$P_{1,4}^1 = 0.3$	$P_{4,3}^1 = 0.4$	$S_4^2 = 2.0$
	$P_{2,3}^1 = 0.5$	$P_{4,5}^1 = 0.6$	$S_5^2 = 6.0$
	$D_1 = \{1,2\}$	$D_4 = \{1,2\}$	$D_5 = \{1,2\}$

Figure 10: Large QN model for a Mixed Queueing Network

Set of Parameters	K_1	K_2	K_3	K_4	K_5	pss	rss	$precision$
exact	10	10	10	8	8	96,059,601	1,353,627	10 digits
A	10	8	9	8	8	71,449,290	1,006,830	4 digits
B	10	7	8	7	8	45,163,008	635,328	5 digits
C	10	7	8	7	8	45,163,008	613,872	6 digits

Table 4: Accuracy of approximation for a two-class mixed FCQN

and the number of correct digits in computing the average length of each queue (*precision*). The first row of this table expresses the solution of the exact equivalent model, *i.e.*, using the original queue capacities of the model. Observing the next three rows, the sensibility of the approximation technique to the low workload demand (low utilization indices) is once more observed. It shows the good compromise between the precision achieved and the quite significant reduction of the problem size expressed by both the product and the reachable state spaces. The absolute reduction of the number of reachable state space benefits the solution by SMART which operates only over this state space. For the solution by PEPS, the reduction of the product state space is more important, since PEPS usually has a mitigated efficiency to handle models with an important amount of non-reachable states.

5 Conclusion

In this paper we presented the principles and the scientific foundations of the MQNA software tool. The results achieved by the use of this software seem to be very promising, since it conjugates the simplicity of QN models with the efficiency and large scope of complex Markovian formalisms like SAN and SMART. One of still inconclusive, but promising, contributions of MQNA software is the possibility to generate smaller models by the approximation technique of reducing the queues capacities. The automatic generation of a myriad of examples through MQNA may also improve the use of the PEPS and SMART software tools and make much easier to compare this two similar modeling approaches.

The future works concerning the MQNA tool may include the exportation for other solvers like GreatSPN [11], and even other formalisms like Stochastic Process Algebras and consequently, to PEPA Workbench software [16]. Additionally, the generation of a structured Petri net model, like Superposed Generalized Stochastic Petri Nets [10], is also a natural future work. In fact, all SPN models currently generated already have a clear structured form (each queue is a set of places and synchronizing transitions).

All these future work ideas may be carried out by a large number of users in the research community, since the MQNA is an open source academic software implemented using only public packages (gcc and Linux). In fact, the beginning of the MQNA project is precisely to integrate, as larger as possible, a number of new technologies in formalisms to model complex systems using the most known formalism in the area, the Queueing Networks.

References

[1] K. Atif and B. Plateau. Stochastic Automata Networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[2] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, 1975.

[3] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In *Performance TOOLS 2003*, Urbana, Illinois, USA, 2003. Springer-Verlag.

[4] B. Beyaert, G. Florin, P. Lonc, and S. Natkin. Evaluation of computer system dependability using Stochastic Petri Nets. In *Proceedings of the* 11^{th} *International Symposium on Fault Tolerant Computing*, pages 24–26, Portland, June 1981.

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 1998.

[6] L. Brenner, P. Fernandes, and A. Sales. MQNA: Markovian Queueing Networks Analyser. http://www.inf.pucrs.br/mqna/.

[7] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. SMART: Stochastic Model Analyzer for Reliability and Timing. In *Tools of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 29–34, Aachen, Germany, September 2001.

[8] Y. Dallery, Z. Liu, and D. Towsley. Equivalence, reversibility, symmetry and concavity properties in fork-join queuing networks with blocking. *Journal of the ACM*, 41(5):903–942, 1994.

[9] Y. Dallery, Z. Liu, and D. Towsley. Properties of fork/join queueing networks with blocking under various operating mechanisms. *IEEE Transactions on Robotics and Automation*, 13(4):503–518, 1997.

[10] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proceedings of the* 15^{th} *International Conference on Applications and Theory of Petri Nets*, pages 258–277.

[11] S. Donatelli, H. Hermanns, J. Hillston, and M. Ribaudo. GSPN and SPA Compared in Pratice. In *Quantitative Methods in Parallel Systems*, pages 38–51.

[12] A. Economou and D. Fakinos. Product form stationary distributions for queueing networks with blocking and rerouting. *Queueing Systems*, 30:251–260, 1998.

[13] P. Fernandes and B. Plateau. Modeling Finite Capacity Queueing Networks with Stochastic Automata Networks. In *Fourth International Workshop on Queueing Networks with Finite Capacity (QNETs 2000)*, pages 1–12, Ilkley, West Yorkshire, UK, July 2000.

[14] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor - Vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[15] E. Gelenbe. Product form queueing networks with negative and positive customers. *Journal of Applied Probability*, 28:656–663, 1991.

[16] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Computer Performance Evaluation*, pages 353–368, 1994.

[17] J. R. Jackson. Networks of waiting lines. *Operations Research*, 5:518–521, 1957.

[18] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, New York, 1975.

[19] M. Ajmone Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli, and G. Franceschinis. An Introduction to Generalized Stochastic Petri Nets. *Microelectronics Reliability*, 31(4):699–725, 1991.

[20] D. A. Menasce and V. A. F. Almeida. *Capacity planning for web services: metrics, models, and methods*. Prentice Hall, 2002.

[21] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, C-31(9):913–917, 1982.

[22] J. L. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3):223–252, 1977.

[23] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[24] B. Plateau. *De l'Evaluation du Parallélisme et de la Synchronisation.* PhD thesis, Paris-Sud, Orsay, 1984.

[25] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):313–322, 1980.

[26] L. D. Servi and S. G. Finn. M/M/1 queues with working vacations (M/M/1/WV). *Performance Evaluation*, 50:41–52, 2002.

[27] S. D. Shapiro. A Stochastic Petri Nets with application to modeling occupancy times for concurrent task systems. *Networks*, 9, 1979.

[28] W. J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

[29] F. J. W. Symons. The description and definition of Queueing Systems by numerical Petri nets. *Australian Telecommunication Research*, 13:20–31, 1980.

[30] D. Towsley. Queueing network models with state-dependent routing. *Journal of the ACM*, 27(2):323–337, 1980.

[31] D. Towsley, P. D. Sparaggis, and C. G. Cassandras. Optimal routing and buffer allocation for a class of finite capacity queueing systems. *IEEE Transactions on Automatic Control*, 37(9):1446–1451, 1992.