DOI 10.1007/s00165-006-0011-6
BCS © 2006
Formal Aspects of Computing (2007) 19: 3–33

Formal Aspects of Computing

Formal techniques for performance analysis: blending SAN and PEPA

Jane Hillston[1] and Leïla Kloul[2]

[1]Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh, Edinburgh EH9 3JZ, UK
[2] PRiSM, Université de Versailles, 45, Avenue des Etats-Unis, 78035 Versailles Cedex, France

Abstract. In this paper we consider two performance modelling techniques from the perspectives of model construction, generation of an underlying continuous time Markov process, and the potential for reduction in the Markov process. Such careful comparison of modelling techniques allows us to appreciate the strengths and weaknesses of different approaches, and facilitates cross-fertilization between them. In the present case we take a characteristic of one formalism, functional rates in Stochastic Automata Networks, and introduce it to the other formalism, Performance Evaluation Process Algebra. We investigate the benefits of this cross-fertilization, particularly from the perspectives of Markov process generation and reduction.

1. Introduction

Performance analysis is the study of system dynamics from the, often conflicting, perspectives of timeliness of behaviour and efficient use of resources. Such study can be carried out by direct experimentation, monitoring and measurement. However, in the domain of computer systems, it is often important that the analysis is carried out before the system is constructed or configured and therefore modelling is widely employed.

Historically, stochastic performance models have been, essentially, informal, as witnessed by the predominance of queueing networks through the 1960s, 1970s and 1980s. Queueing networks give a very compact representation of systems with resource contention between independent customers. Moreover, analytical methods and well known algorithms may be used to obtain either analytical or numerical results. However, whilst conventions exist, there is no formal interpretation: correct analysis and solution of models rely on the expertise of the modeller both in terms of the notation and the system under study.

The evolution of computer systems in the last twenty years has introduced several features which are difficult to capture in queueing networks. In particular, queueing networks are inefficient whenever complex synchronisation constraints need to be taken into account. Addressing these concerns, a new generation of performance modelling techniques have been introduced over the last two decades. The majority of these seek to provide a high level model description technique from which a continuous time Markov chain can be derived as the underlying stochastic model. Since this has coincided with the growth of "formal methods" for system specification it is unsurprising that many of these new techniques have been formed by incorporating stochastic information into existing formal methods. Thus now we can consider there to be two broad classes of performance modelling techniques: those which seek to address the shortcomings of queueing networks but which retain the informal characteristic, and those which are based on incorporation of stochastic information into existing formal methods.

Correspondence and offprint requests to: Leïla Kloul, PRiSM, Université de Versailles 45, Avenue des Etats-Unis 78035 Versailles Cedex, France. E-mail: kle@prism.uvsq.fr

In the former category, for example, we can include Plateau's Stochastic Automata Networks (SANs) [Pla85] and various queueing network extensions e.g. [HPTvD90, HT90, JL82]. There are many examples of formal stochastic performance modelling approaches in the latter category. For example, stochastic Petri nets [Mol82] and generalized stochastic Petri nets [ACB84], Stochastic Activity Networks [SM91] and stochastic process algebras [Hil94, BG98, GHR93, Her99, DHKK01]. In all cases the objective is to have a high-level modelling formalism from which it is possible to automatically generate the underlying continuous time Markov chain. This stochastic model is characterised by a matrix termed the *generator matrix*.

Stochastic process algebras emerged as a new performance modelling technique in the early 1990s. Several languages have been published and these can be broadly categorised into those in which activity and time/delay are combined and those in which they are treated as orthogonal. The stochastic process algebra which we consider, Hillston's Performance Evaluation Process Algebra (PEPA) [Hil94], falls into the former category. PEPA extends classical process algebra by associating a random variable, representing duration, with every action. These random variables are assumed to be exponentially distributed and this leads to a clear mapping from the process algebra model to the continuous time Markov process.

This can be regarded as a fully fledged formal method, since the language includes a calculus for model manipulation and analysis, based on formally defined equivalence relations. Considerable effort has been applied to studying the relationship between the process algebra structure and techniques and the underlying Markov process, and the extent to which the former can be exploited in the latter [Hil05b]. For example, in PEPA there is a model reduction technique, termed *aggregation*, which can be used to reduce the state space of the Markov process via a behaviour preserving equivalence to partition the state space at the process algebra level [Hil95].

PEPA is supported by several software tools including PRISM [KNP02] and the PEPA Workbench [CGHT99], which incorporates procedures to carry out automatic aggregation [GHR01].

Unlike PEPA, the SAN modelling approach is best described as *semi-formal* since it is based on automata theory but relies on informal annotation by the modeller. SANs are based on graphical models in conjunction with an underlying stochastic hypothesis. The dynamic behaviour of a component is represented by a transition from one state to another; each state is represented by a node, each transition by an arc between states. A label on the transition specifies the instant and the occurrence probability of the action. This label specifies also the type of the action. Again, the objective is to derive an underlying continous time Markov process. The informal annotations are required in order to make the specification complete with respect to this stochastic process. There appear to be some moves towards making SAN more formal since recent changes [BBF$^+$03] to the PEPS tool which supports the approach, oblige the modeller to declare explicitly the automata states and the network events.

The principal strength of SAN lies in their efficient solution technique which avoids the construction of the monolithic generator matrix representing the underlyng Markov process. Under adequate probabilistic hypotheses, the behaviour of a SAN is represented by a multi-dimensional Markov process whose states are those of the product space. Using a technique based on *tensor* or *Kronecker* algebra, it has been proved [Pla85, PF91] that this method automatically provides an analytic derivation of a decomposed form of the generator matrix called the *descriptor*. Compared to a monolithic description of the generator, the structure of this descriptor leads to a considerable reduction in memory requirements during model solution. Moreover, solution techniques have been adapted to this representation [FPA98].

In this paper, after presenting the strengths and weaknesses of each of the formalisms, we propose an enhanced version of PEPA in which we adopt a system description feature of SAN which was not previously available to PEPA modellers. SAN allows functional dependencies to be expressed between automata, which mean that the timing behaviour of one component (or indeed the activities it is able to undertake) may depend on the current state of another component. It is this feature that we introduce into PEPA. This introduces some modelling flexibility into PEPA and in some cases can lead to a reduction of the model's size. Furthermore, we study its impact on the current state space reduction techniques based on bisimulation.

Contribution of this paper We present two contrasting approaches to performance analysis and demonstrate the benefits which can be gained by importing a characteristic of one formalism, functional dependencies, into the other. Whilst the use of functional dependencies is well-established in untimed process algebras, this is novel in the context of a stochastic process algebra used for performance modelling. Moreover, we focus on the benefits of this extension with respect to the expression and solution of the underlying Markov chain. By necessity when working at this level of detail we focus on the particulars of the two chosen formalisms, SAN and PEPA, but these may be regarded as representative of their respective wider classes of performance modelling approaches. Thus the current work serves to illustrate the benefits which may be gained by cross-fertilisation between formalisms and from developing a single framework in which multiple model reduction techniques can be studied and compared.

Structure of the paper Some related work which helps to establish the context of the current paper is presented in Sect. 2 before Sect. 3 briefly introduces the PEPA and SAN formalism, and demonstrates their modelling style on a small example. Section 4 is devoted to a case study in which we show the interest of the functional dependencies. In Sect. 5 we show how to introduce these dependencies in PEPA. Section 6 explains the methods used to tackle the state space explosion problem in SAN and PEPA and Sect. 7 considers the impact of the introduction of functional rates and explains how a tensor representation for PEPA models can be developed. Using a second case study, we show, in the first part of Sect. 8, how to build the different matrices required for the descriptor of PEPA models. In the second part of the same section we show how to apply the simplification technique on the PEPA model. We finally conclude with some remarks and possible future works.

2. Related work

Stochastic process algebras (SPAs) emerged in the early 1990s as a performance modelling formalism. In contrast to classical process algebras they incorporate probabilistic choice and random timings, although in many cases the choice is implicit as in PEPA, arising from the race condition between different timed actions. For SPAs supporting synchronised communication the timing may be incorporated in the actions as in PEPA, EMPA, TIPP and Stochastic π-calculus [Hil94, BG98, GHR93, Pri95], or regarded as orthogonal to it as in IMC and MoDeST [Her99, DHKK01]. There are also process algebras in which communication is via asynchronous message passing and these have also recently been given stochastic extensions [DLM05, DHW04, Bor06].

In this paper we are advocating the usefulness of detailed comparisons between formalisms, and if appropriate, the extension of one formalism with characteristics of another. Previously, the relationship between stochastic Petri nets and stochastic process algebras has been extensively studied from both a pragmatic and a theoretical perspective [Rib95, DHR95, DHHR95, HRRS01]. That study has resulted in fruitful cross-fertilization between the formalisms [GHKR03].

The characteristic we import from SAN to PEPA is functional dependencies. Such dependencies have previously been used in untimed process algebras such as μCRL [GP95] and LOTOS [ISO98]. However our particular focus here is that the functional dependent is a *rate* and we explore the implications of this with respect to the generation and solution of the continuous time Markov chain underlying the stochastic process algebra model. We believe that this is the first time that functional dependencies have been developed for a synchronising stochastic process algebra, whilst a proposal for the asynchronous interaction stochastic process algebra SCCP has been made by Bortolussi contemporaneously [Bor06].

One of the consequences of introducing functional rates into PEPA is that it facilities a tensor, or Kronecker, representation of the underlying Markov chain. Kronecker algebra representations have been used for some time as a means to address the state space explosion problem arising in the numerical solution of Markov chains. The pioneering work in this area was carried out by Plateau in the context of SANs [Pla84].

More recently, Kronecker-based solution techniques have been developed for various Petri net based-formalisms. In [Don94], the approach developed is based on finding a partition of the GSPN. The resulting GSPNs synchronize on timed transitions. In this approach no immediate transitions are allowed and firing the synchronizing transitions must lead to tangible states. Whilst the approach used in [Kem96] is similar, the class of GSPN considered is more general; the constraint on the firing the synchronizing transitions is relaxed. Moreover, Kemper presents an algorithm which generates the reachability set which profits from the Kronecker form. In [CM96], no particular structure of the net is assumed and a more general marking behaviour is allowed. The approach developed by the authors consists of obtaining the Kronecker representation for individual places, which results in small matrices, and then compacting the places into "macro-places" which correspond to the sub-GSPNs. The final expression for the places contains a matrix inverse which cannot be expressed using Kronecker operators on small matrices. This disappears when no immediate transitions are considered. Moreover, the size of the transition matrix is enormous, potentially leading to inefficiency because of the large number of unreachable states. In a more recent work [DK01], synchronisation with priority has been integrated into a Kronecker representation of GSPNs.

In all these works, the Kronecker representations of the generator matrix do not avoid the unreachable states. In [Buc99], an approach which avoids the unreachable states of the Markov chain is investigated. However, this approach seems to be incompatible with the standard solution algorithms because of the structure of the matrix.

In [CM99], the data structure to store the reachable states and the generator matrix is studied. A matrix diagram which is a combination of Kronecker and BDD-based approach is used to store the generator.

With their explicit compositional structure, SPAs would appear to be natural candidates for Kronecker representation. In 1994 Buchholz proposed an SPA called *MPA*, for which the mapping to an underlying Markov process is only defined in terms of a tensor expression [Buc94]. However, in MPA the interpretation of both basic actions and shared actions is quite different to that in PEPA, as it was chosen specifically to facilitate the tensor representation and without a natural modelling interpretation.

The tensorial representation of the Markov process underlying a PEPA model has some similarities to those previously developed for SAN and SPN; nevertheless it also has novel features. In particular capturing the correct timing behaviour of cooperating PEPA activities relies on functional dependencies.

3. PEPA and SAN

In this section we give brief introductions to each of the formalisms we consider, and then informally compare them on the basis of small examples. More details of PEPA can be found in [Hil94], whilst more details of SAN can be found in [Pla85].

3.1. PEPA

The basic elements of PEPA are *components* and *activities*, corresponding to *states* and *transitions* in the underlying Markov process. Each activity has an *action type* (or simply *type*). Activities which are private to the component in which they occur are represented by the distinguished action type, τ. The duration of each activity is represented by the parameter of the associated exponential distribution: the *activity rate* (or simply *rate*) of the activity. This parameter may be any positive real number, or the distinguished symbol $\top$ (read as *unspecified*). Thus each activity, a, is a pair (α, r) where α is the action type and r is the activity rate. We assume that there is a countable set of components, which we denote $\mathcal{C}$, and a countable set, $\mathcal{A}$, of all possible action types. We denote by $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^+$, the set of activities, where $\mathbb{R}^+$ is the set of positive real numbers together with the symbol $\top$. The combinators, together with their names and interpretations, are presented informally below; the structured operational semantic rules for the language are included in Appendix A.

Prefix: $(\alpha, r).P$ Prefix is the basic mechanism by which the behaviours of components are constructed. The component carries out activity (α, r) and subsequently behaves as component P.

Choice: $P + Q$ The component represents a system which may behave either as component P or as Q: all the current activities of both components are enabled. The first activity to complete, determined by a *race condition*, distinguishes one component, the other is discarded. The choice combinator represents competition between components.

Cooperation: $P \underset{L}{\bowtie} Q$ The components proceed independently with any activities whose types do not occur in the *cooperation set* L (*individual activities*). However, activities with action types in the set L require the simultaneous involvement of both components (*shared activities*). These activities are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. Thus one component may become blocked, waiting for the other component to be ready to participate. The shared activity occurs at the rate of the slowest participant. If an activity has an unspecified rate in a component, the component is *passive* with respect to that action type. This means that the component does not influence the rate at which any shared activity occurs. The cooperation combinator associates to the left but brackets may also be used to clarify the meaning. When the set L is empty, we use the more concise notation $P \parallel Q$ to represent $P \underset{\emptyset}{\bowtie} Q$.

Hiding: P/L The component behaves as P except that any activities of types within the set L are *hidden*, i.e. such an activity exhibits the unknown type τ and the activity can be regarded as an internal delay by the component. Such an activity cannot be carried out in cooperation with any other component: the original action type of a hidden activity is no longer externally accessible, to an observer or to another component; the duration is unaffected.

Constant: $A \overset{def}{=} P$ Constants are components whose meaning is given by a defining equation: $A \overset{def}{=} P$ gives the constant A the behaviour of the component P. This is how we assign names to components (behaviours). There is no explicit recursion operator but components of infinite behaviour may be readily described using sets of mutually recursive defining equations.

From the SOS rules (see Fig. 8 in the Appendix) each PEPA model can be regarded as a labelled *multi*-transition system $(\mathcal{C}, \mathcal{A}ct, \{\!| \xrightarrow{(\alpha,r)} \mid (\alpha, r) \in \mathcal{A}ct\ |\!\})$ where $\mathcal{C}$ is the set of components, $\mathcal{A}ct$ is the set of activities and the multi-relation $\xrightarrow{(\alpha,r)}$ is given by the rules. Note that the multiplicity is important in order to keep the stochastic information contained in the model intact. This transition system, or *derivation graph*, also characterises the Markov process represented by the model. The nodes of the graph (states of the Markov process) are the syntactic terms exhibited by the model, and arcs represent the possible transitions between them.

Necessary (but not sufficient) conditions for the ergodicity of the Markov process in terms of the structure of the PEPA model have been identified and can be readily checked [Hil94, GHR97]. Ergodicity implies that the model must give rise to a strongly connected derivation graph. If we consider the layering imposed on a component by cooperation combinators, this implies that choice combinators may only be introduced at the lowest level of a cyclic component since syntactic terms are associated with states.

This leads us to formally define the syntax of PEPA expressions in terms of *sequential components* S and *model components* P:

$$P ::= S \mid P \underset{L}{\bowtie} P \mid P/L$$
$$S ::= (\alpha, r).S \mid S + S \mid A$$

3.2. Stochastic Automata Networks

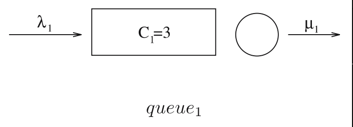
In the SAN approach, a system is represented by a number of *automata*, each automaton capturing the dynamic behaviour of an element of the system. These elements are assumed to work more or less independently of one another, with only limited interaction. Within an individual automaton, the behaviour of a component or aspect of the system is captured as a set of *states* and rules that govern the *events* causing movement from one state to another. Automata are defined in terms of state transition diagrams in which each node represents a state and each arc represents a state change. Arcs are labelled in such a way as to capture information about the rate at which an event occurs and possibly synchronisation between events [PFL88]. Thus transitions can be of two types: *local* or *synchronised*. A local transition occurs only in the automaton whereas a synchronised transition occurs in several automata at the same time.

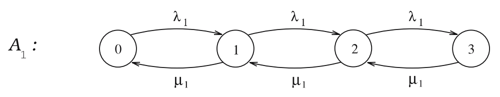
More formally, a Stochastic Automata Network (SAN) is a set of N automata in which each automaton A_i is defined by the tuple (S_i, L, Q_i) where

- S_i is the set of states of the automaton,
- L is the set of labels. A label l is a list that may contain either a function τ, or a list of tuples (e, τ_e, p_e) with different symbols e, or both of them such that:
  - e is the name of the synchronising event or synchronisation,
  - τ and τ_e are the transition rates, functions defined from $\Pi_{i=1}^{N} S_i$ to $\mathbb{R}^+$,
  - p_e is the probability transition function defining a conditional routing probability between local states given that the synchronising event occurs.
- Q_i is the transition function which associates a label from L with every arc of A_i.

A label on an edge allows us to specify the type and the rate of the transition as follows:

- If $Q_i(x_i, y_i)$ contains a function τ, then we have a transition local to automaton A_i between states x_i and y_i. If τ is not a constant, the transition is still local to automaton A_i, but its rate depends on the state of the other automata of the network.
- If $(e, \tau_e, p_e(x_i, y_i)) \in Q_i(x_i, y_i)$, then the transition between states x_i and y_i is a synchronised transition, e and τ_e being the name and the rate of the transition of the synchronising event. $p_e(x_i, y_i)$ is the routing probability between local states x_i and y_i. The name of the event allows us to define the automata and the transitions concerned by this event. The distinction between the rate and the probability is required because the first must be





The SAN model

$$
\begin{aligned}
Buffer_0 &\stackrel{def}{=} (in, \lambda_1).Buffer_1\\
Buffer_1 &\stackrel{def}{=} (in, \lambda_1).Buffer_2 + (service, \mu_1).Buffer_0\\
Buffer_2 &\stackrel{def}{=} (in, \lambda_1).Buffer_3 + (service, \mu_1).Buffer_1\\
Buffer_3 &\stackrel{def}{=} (service, \mu_1).Buffer_2
\end{aligned}
$$

The PEPA model

Fig. 1. A single buffer system model

unique for a given synchronising event, thus the same on all concerned automata i.e. the rate of synchronising transitions are determined at the global level, whilst the probabilities may (and generally will) differ.

Thus synchronising transitions, which are made up of events from two or more automata, provide a direct form of interaction between automata in a SAN. However, automata are also able to influence each other in a less direct way using the mechanism of functional rates. It is possible for the rate of a local transition within one automaton to be influenced by the local states of one or more other automata. Both these forms of interaction will be illustrated by the example in the following subsection.

3.3. High-level model abstractions for representation

We begin our investigation of the relationship between the two formalisms by considering how they model systems, i.e. the high-level model abstractions used in each case. We do this via a small example: a single buffer system, as depicted in the left-hand side of Fig. 1. This is a buffer *queue*$_1$ with three places (capacity $C_1 = 3$). When the buffer is empty only an arrival is possible. When the buffer is full arrivals are suspended and only a departure, representing the completion of some service, may occur. In other states an arrival or a departure may occur. We assume that the arrival rate is λ_1 and the departure rate is μ_1. This example will also serve to clarify the descriptions of SAN and PEPA given in the previous sections.

Both SAN and PEPA aim to provide a compositional framework for model construction, allowing the modeller to focus on the behaviour of individual components within a system and the interactions between them rather than tackling the complexity of the whole system all at once.

Events vs activities Using SAN the behaviour of the buffer is modelled using an automaton $\mathcal{A}_1$ composed of four states $(0, \ldots, 3)$, each one corresponding to the number of customers that may be present in the buffer after the occurrence of an event. In this single buffer system, the events that may occur are the arrival of a new customer and the service completion. In automaton $\mathcal{A}_1$ (Fig. 1), the effect of the first event is represented by the transitions with rate λ_1 and the effect of the second by the transitions with rate μ_1.

In PEPA, the buffer would be represented by a model composed of one sequential component *Buffer*$_0$ with three derivatives *Buffer*$_i$ where i, $i = 1..3$, is the number of customers in the buffer.[1] We consider two activities with types *in* and *service*. The first one is used to describe the arrival of a new customer in the buffer and the second the service completion. As we can see, only the prefix and choice combinators are used in the construction of sequential components (Fig. 1). When the buffer is empty, only activity *in* is enabled, whereas when the buffer is full, only activity *service* can be performed. However, in all other cases both activities are enabled since we can have *either* the arrival of a new customer or the service completion of one in the buffer.

[1] Note that the constants used to distinguish the derivatives are given suggestive names, e.g. *Buffer*$_i$ when there are i customers in the buffer, but they are not formally parameterised.

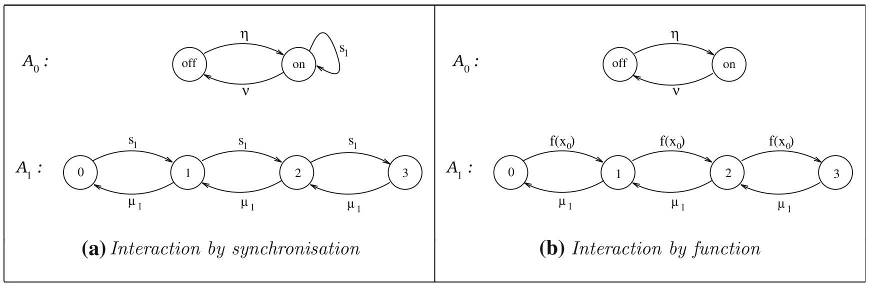

A_0 :
off
on
η
ν
s_1
A_1 :
0
1
2
3
s_1
μ_1
(a) Interaction by synchronisation
$f(x_0)$
(b) Interaction by function

Fig. 2. Automata interactions

Based on this simple example, we can see that PEPA may be regarded as a more explicit formalism than SAN. In PEPA, each part of the behaviour of a system is explicitly modelled. In SAN models, the system behaviour is implicitly represented since the transitions are the result of the implicit occurrence of events. The following will confirm this observation.

Components interaction Now we consider the single buffer system described above but assume that it accepts arrivals from a distinct arrival process which is a simple *on/off* source. The system must be represented as the interaction of two components, the source and the buffer.

In a SAN model of the system we use two automata $\mathcal{A}_0$ and $\mathcal{A}_1$, the first to model the source behaviour and the second to model $queue_1$ (Fig. 2 (a)). The events that may occur in such a system are the following:

1. The source switches to state *on*. This event is represented by the transition with rate η in automaton $\mathcal{A}_0$.
2. The generation of a customer by the source and its arrival in $queue_1$. As this event concerns both the source and $queue_1$, this is represented by synchronisation transitions labelled s_1 on both automata. This synchronisation is at rate λ_1 and its routing probability is 1. Note that when it is not specified on the automaton, it is equal by default to 1.
3. The source switches to state *off*. This event is represented by the transition with rate ν in automaton $\mathcal{A}_0$.
4. The service completion of a customer in $queue_1$. It is represented by the transitions with rate μ_1 on automata $\mathcal{A}_1$.

Events 1, 3 and 4 concern only one automaton. These events are *local events* whereas event 2 is a *synchronising event*.

We can also illustrate the less direct form of interaction in SAN, based on functional rates, using the same example as above but taking a different representation of the arrival process. Now we capture only whether the process is currently "*on*" or not. So, the transitions in automaton $\mathcal{A}_1$ which capture the arrival of a new customer are no longer synchronising transitions, but *functional transitions* (Fig. 2 (b)). If variable x_0 denotes the local state of automaton $\mathcal{A}_0$ then we can define a function f as

$$f(x_0) = \begin{cases} 1 & \text{if } x_0 = on \\ 0 & \text{otherwise} \end{cases}$$

The rate of a transition labelled $f(x_0)$ depends on the state of $\mathcal{A}_0$, i.e. there is a *functional dependency* between the two automata.

Note that in SAN there is little sense of identity within the model. Individual states and events are not *named* in any explicit sense. Identities are only explicitly associated with automata and synchronisations. However, when a functional rate is used, as in the example above, there is an implicit assumption that the local states of the influencing automaton are labelled, although this labelling need not be unique. This is necessary for the definition of the function which determines the transition rate in the influenced automaton.

In contrast, in PEPA, only one form of component interaction is allowed, that provided by *cooperation*. Similar to synchronising events in SAN, activities carried out in cooperation require the synchronous participation of two or more components. For example, consider the system composed of the three place-buffer and the *on/off* source. To model this system we need two components $Buffer_0$ and $Arrival_{on}$. Thus, when we wish to model the arrival process to the buffer separately, with the *in* activity shared, the components are defined as

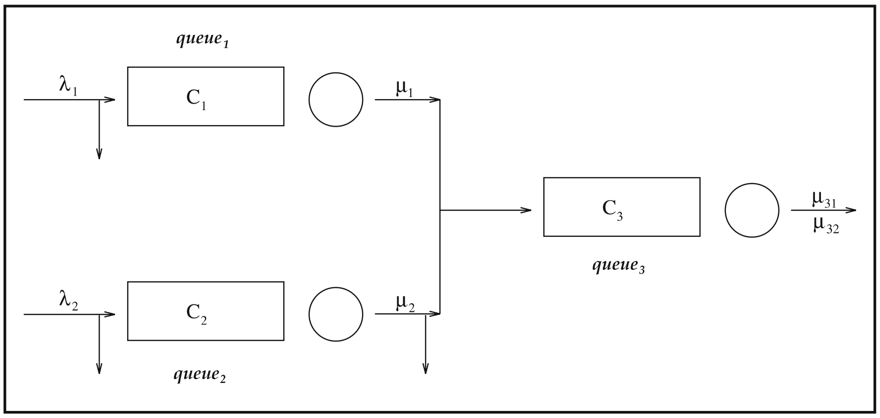


Fig. 3. A finite capacity queueing network

shown below.

$$\begin{aligned}
Buffer_0 &\overset{def}{=} (in, \top).Buffer_1 & Arrival_{on} &\overset{def}{=} (in, \lambda).Arrival_{on} \\
& & &\quad +(off, \nu).Arrival_{off} \\
Buffer_1 &\overset{def}{=} (in, \top).Buffer_2 + (service, \mu).Buffer_0 & Arrival_{off} &\overset{def}{=} (on, \eta).Arrival_{on} \\
Buffer_2 &\overset{def}{=} (in, \top).Buffer_3 + (service, \mu).Buffer_1 & & \\
Buffer_3 &\overset{def}{=} (service, \mu).Buffer_2 & &
\end{aligned}$$

The rate of activity *in* is undefined ($\top$) in component $Buffer_0$ since the value of this rate depends on the state of component $Arrival_{on}$ in which it is specified. Moreover, as this activity requires the synchronous participation of both components $Buffer_0$ and $Arrival_{on}$, in addition to the mutually recursive sets of equations defining the behaviour of each sequential component, we also need a *system equation* which defines the cooperation between the two components:

$$System \overset{def}{=} Buffer_0 \underset{\{in\}}{\bowtie} Arrival_{on}$$

The set decorating the cooperation combinator $\bowtie$ specifies which activities require the synchronous participation of the two arguments of the combinator, in this case $Buffer_0$ and $Arrival_{on}$. Other activities of the two components, such as *service* and *off* can be carried out independently.

In PEPA the cooperation is the only form of component interaction. There is no mechanism for setting *functional rates* for PEPA activities, analogous to that in SAN. In the following, we show, using a case study, that these functional dependencies provide the SAN modeller with a certain flexibility in building their model which would be advantageous to introduce into the PEPA formalism.

4. Case study 1: a finite capacity queueing network

Consider an open queueing network composed of three finite capacity queues called $queue_1$, $queue_2$ and $queue_3$ (Fig. 3). We denote by C_i the finite buffer capacity at $queue_i$, $i = 1 \ldots 3$. The customers of this network are of two classes. Class 1 customers arrive from outside the network to $queue_1$ according to a Poisson process with rate λ_1. An arriving customer is considered lost if $queue_1$ is full when it arrives. Similarly, class 2 customers arrive from outside the network to $queue_2$ according to a Poisson process with rate λ_2. If $queue_2$ is full when a class 2 customer arrives, it is considered lost also. The servers at $queue_1$ and $queue_2$ provide exponential service at rate μ_1 and μ_2 respectively.

After its service in $queue_1$, a class 1 customer tries to join $queue_3$. If this buffer is full, the customer is blocked in the server until a slot becomes available in $queue_3$. Similarly, on its service completion in $queue_2$, a class 2 customer tries to join $queue_3$. However if the buffer is full, the class 2 customer is discarded.

The server at $queue_3$ provides an exponential service at rate μ_{31} to class 1 customers and at rate μ_{32} to class 2 customers. We assume that class 1 customers have a preemptive priority over class 2 customers for the service. Customers completing service at $queue_3$ leave the network.

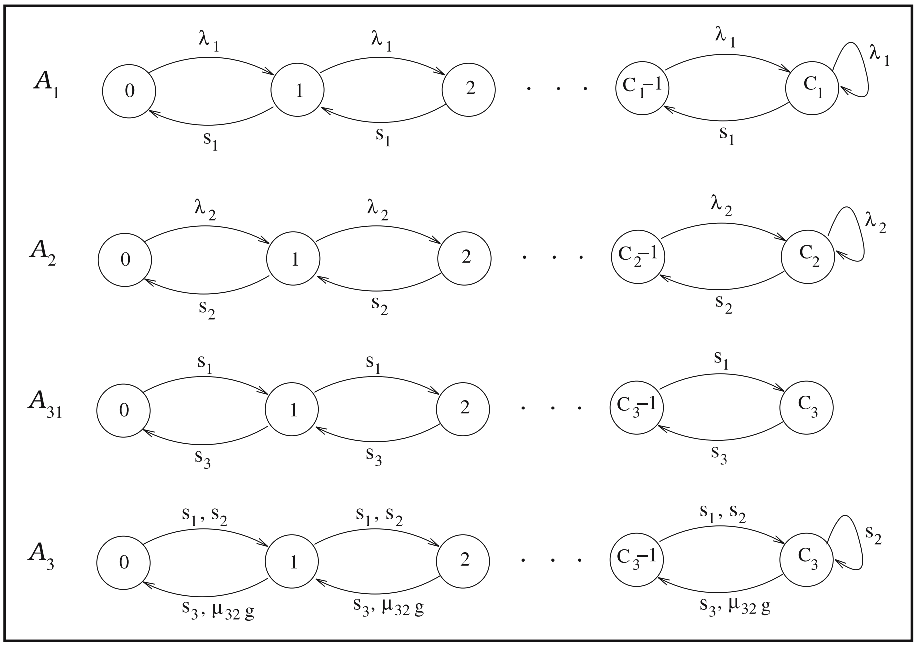


Fig. 4. The SAN Model 1

In the following we show how to model this system using both our chosen formalisms. When presenting the SAN model we will show that there are alternatives, made possible by the use of functional rates, which do not exist for the PEPA model.

4.1. The SAN models

To capture the behaviour of the whole network, we need to capture the behaviour of each of the queues and the interactions between them. We model $queue_1$ and $queue_2$ using automaton $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, and establish a one-to-one correspondence between the number of customers in the buffer and the state of the associated automaton. Representing $queue_3$ requires two automata because the customers in its buffer are of two classes and one class has a preemptive priority service over the other. In our model the first automaton ($\mathcal{A}_{31}$) provides the number of class 1 customers in $queue_3$, whereas the second ($\mathcal{A}_3$) provides the total number of customers of both classes (Fig. 4).

To understand the different automata of the SAN model presented in Fig. 4 let us consider what are the events that may occur in the corresponding queueing network.

1. The arrival of a new customer in $queue_1$ ($queue_2$). This is a local event since it has no effect on the other queues of the network. It is represented by a local transition in automaton $\mathcal{A}_1$ ($\mathcal{A}_2$) with rate λ_1 (λ_2).
2. The service completion in $queue_1$. Either there is an available slot in $queue_3$ and the customer joins it, or the buffer is full and therefore the customer is blocked in the server of $queue_1$. In both cases, both $queue_1$ and $queue_3$ are involved by this event. This is modelled using synchronisation s_1 which appears in automata $\mathcal{A}_1$, $\mathcal{A}_3$ and $\mathcal{A}_{31}$ with rate μ_1.
3. The service completion in $queue_2$. In this case, either there is an available slot in $queue_3$ and the customer joins it, or the buffer is full and then the customer is lost. This event is represented using synchronisation s_2 with rate μ_2 in automata $\mathcal{A}_2$ and $\mathcal{A}_3$. The case of the loss of the customer is represented by the loop on the last state of $\mathcal{A}_3$.
4. The service completion in $queue_3$. If the customer is of class 1 this event is represented using synchronisation s_3 with rate μ_{31} in automata $\mathcal{A}_{31}$ and $\mathcal{A}_3$. The service completion of a class 2 customer is represented by the transitions in $\mathcal{A}_3$ with the functional rate $\mu_3\, g(x_{31})$ where $g(x_{31}) = 0$ if there is at least one class 1 customer in $queue_3$, otherwise $g(x_{31}) = 1$.

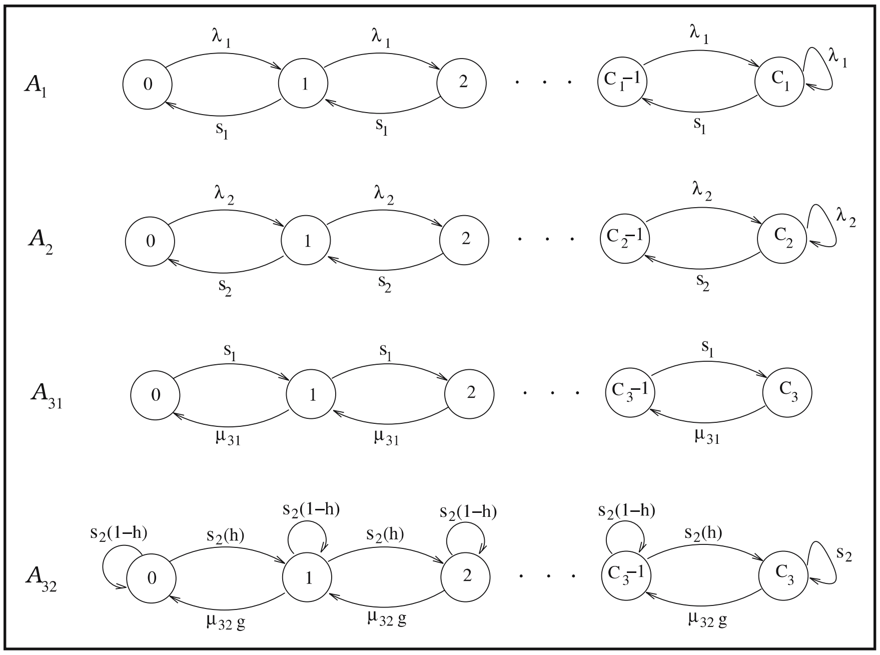


Fig. 5. The SAN Model 2

The functional rates in SAN permit alternative models of this system to be developed. In the model described above (Fig. 4), *queue*$_3$ is modelled considering two aspects of this buffer: the total number of class 1 customers and the total number of customers. An alternative is to consider the total number of class 1 customers and the total number of class 2 customers in the buffer. The corresponding SAN model is given in Fig. 5.

As in the first model, automata $\mathcal{A}_1$ and $\mathcal{A}_2$ model the total number of customers in *queue*$_1$ and *queue*$_2$ respectively. $\mathcal{A}_{31}$ provides the total number of class 1 customers in *queue*$_3$ and $\mathcal{A}_{32}$ provides the total number of class 2 in the same queue.

The events in this model are as listed above. The external arrivals and service completion at *queue*$_1$ are represented as previously in the automata concerned. The other events are represented as follows:

- The service completion in *queue*$_2$. This event is represented using synchronisation s_2 with rate μ_2 in automaton $\mathcal{A}_2$. In automaton $\mathcal{A}_{32}$, the rate is μ_2 with the routing probability $h(x_{31})$ for the transitions from state i to $i+1$. Here h is a function defined as $h(x_{31}) = 1$ if the number of class 1 customers in *queue*$_3$ is less than C_3. Otherwise this function is null. The service completion of a class 2 customer and its loss is represented by the loops on the states with a routing probability $1 - h(x_{31})$ for all states except for state C_3 where it is by default equal to 1.
- The service completion in *queue*$_3$. In this model, as we represent the customer number of each class in *queue*$_3$, we have to make a distinction between the service completion of both classes. The service completion of a class 1 customer is represented by a transition with rate μ_{31} in automaton $\mathcal{A}_1$. The service completion of class 2 customer is represented by a transition with rate $\mu_{32}\,g(x_{31})$, g being the same function as in the first model.

4.2. The PEPA model

The PEPA model is composed of four components $Buffer_0^{(1)}$, $Buffer_0^{(2)}$, $Buffer_0^{(3)}$ and $Class1_0^{(3)}$. The two first components describe the behaviour of *queue*$_1$ and *queue*$_2$ respectively. As in the SAN model, two components are required to describe the behaviour of *queue*$_3$; $Buffer_0^{(3)}$ describes the total number of customers of both classes and $Class1_0^{(3)}$ the number of class 1 customers in *queue*$_3$.

Component $Buffer_0^{(1)}$: To capture the behaviour of *queue*$_1$, we need to use four activities types: in_1 describes the arrival of a new customer in the buffer, $service_1$ models a customer's service, $blocking$ represents the blocking after service possibility and finally $loss_1$ models the case where the buffer is full and therefore an arriving customer is lost. The complete behaviour of component $Buffer_0^{(1)}$ is as follows:

$$\begin{array}{lcl}
Buffer_0^{(1)} & \stackrel{def}{=} & (in_1, \lambda_1).Buffer_1^{(1)} \\
Buffer_1^{(1)} & \stackrel{def}{=} & (in_1, \lambda_1).Buffer_2^{(1)} + (service_1, \mu_1).Buffer_0^{(1)} + (blocking, \mu_1).Buffer_1^{(1)} \\
\vdots & & \vdots \\
Buffer_{C_1}^{(1)} & \stackrel{def}{=} & (loss_1, \lambda_1).Buffer_{C_1}^{(1)} + (service_1, \mu_1).Buffer_{C_1-1}^{(1)} + (blocking, \mu_1).Buffer_{C_1}^{(1)}
\end{array}$$

Component $Buffer_0^{(2)}$: This component is similar to the one above with the activities in_2, $service_2$ and $loss_2$ playing analogous roles. Additionally, $loss_2'$ represents the loss of a class 2 customer, when it tries to join $queue_3$, whose buffer is full. The complete behaviour of component $Buffer_0^{(2)}$ is as follows:

$$\begin{array}{lcl}
Buffer_0^{(2)} & \stackrel{def}{=} & (in_2, \lambda_2).Buffer_1^{(2)} \\
Buffer_1^{(2)} & \stackrel{def}{=} & (in_2, \lambda_2).Buffer_2^{(2)} + (service_2, \mu_2).Buffer_0^{(2)} + (loss_2', \mu_2).Buffer_0^{(2)} \\
\vdots & & \vdots \\
Buffer_{C_2}^{(2)} & \stackrel{def}{=} & (loss_2, \lambda_2).Buffer_{C_2}^{(2)} + (service_2, \mu_2).Buffer_{C_2-1}^{(2)} + (loss_2', \mu_2).Buffer_{C_2-1}^{(2)}
\end{array}$$

Component $Buffer_0^{(3)}$: This component describes the behaviour of $queue_3$ for both customer classes. The arrival of a customer to this queue means the end of service either in $queue_1$ (activity $service_1$) or in $queue_2$ (activity $service_2$). These activities are synchronising activities whose rate in this component is unspecified. The service completion of a customer in $queue_3$ is modelled using activity $service_{31}$ or $service_{32}$ according to the class of the customer. Component $Buffer_0^{(3)}$ must synchronise with $Buffer_0^{(1)}$ on the activity $blocking$ when $queue_3$ is full and with $Buffer_0^{(2)}$ on activity $loss_2'$. The complete behaviour of component $Buffer_0^{(3)}$ is as follows:

$$\begin{array}{lcl}
Buffer_0^{(3)} & \stackrel{def}{=} & (service_1, \top).Buffer_1^{(3)} + (service_2, \top).Buffer_1^{(3)} \\
Buffer_1^{(3)} & \stackrel{def}{=} & (service_1, \top).Buffer_2^{(3)} + (service_2, \top).Buffer_2^{(3)} + (service_{31}, \mu_{31}).Buffer_0^{(3)} \\
 & + & (service_{32}, \mu_{32}).Buffer_0^{(3)} \\
Buffer_2^{(3)} & \stackrel{def}{=} & (service_1, \top).Buffer_3^{(3)} + (service_2, \top).Buffer_3^{(3)} + (service_{31}, \mu_{31}).Buffer_1^{(3)} \\
 & + & (service_{32}, \mu_{32}).Buffer_1^{(3)} \\
\vdots & & \vdots \\
Buffer_{C_3}^{(3)} & \stackrel{def}{=} & (service_{31}, \mu_{31}).Buffer_{C_3-1}^{(3)} + (service_{32}, \mu_{32}).Buffer_{C_3-1}^{(3)} + (loss_2', \top).Buffer_{C_3}^{(3)} \\
 & + & (blocking, \top).Buffer_{C_3}^{(3)}
\end{array}$$

Component $Class1_0^{(3)}$: This component models the total number of class 1 customers in $queue_3$. The arrival of a class 1 customer to $queue_3$, means service completion in $queue_1$. This is modelled here by activity $service_1$ on which this component must synchronise with components $Buffer_0^{(3)}$ and $Buffer_0^{(1)}$. When a class 1 customer is served in $queue_3$, activity $service_{31}$ is performed. A class 2 customer is served in this queue only if no class 1 customers are in the buffer. This is modelled by activity $service_{32}$ in $Class1_0^{(3)}$. Below is the complete definition of component $Class1_0^{(3)}$.

$$\begin{array}{lcl}
Class1_0^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_1^{(3)} + (service_{32}, \top).Class1_0^{(3)} \\
Class1_1^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_2^{(3)} + (service_{31}, \top).Class1_0^{(3)} \\
\vdots & & \vdots \\
Class1_{C_3}^{(3)} & \stackrel{def}{=} & (service_{31}, \top).Class1_{C_3-1}^{(3)}
\end{array}$$

The complete system: The complete behaviour of the system is given by the following equation:

$$System \stackrel{def}{=} \left(\left(Buffer_0^{(1)} \underset{\mathcal{K}}{\bowtie} Buffer_0^{(3)} \right) \underset{\mathcal{L}}{\bowtie} Buffer_0^{(2)} \right) \underset{\mathcal{M}}{\bowtie} Class1_0^{(3)}$$

where $\mathcal{K} = \{service_1, blocking\}$ is the cooperation set on which $Buffer_0^{(1)}$ and $Buffer_0^{(3)}$ must synchronise, $\mathcal{L} = \{service_2, loss'_2\}$ the cooperation set on which $Buffer_0^{(3)}$ and $Buffer_0^{(2)}$ must synchronise and finally $\mathcal{M} = \{service_1, service_{31}, service_{32}\}$ is the set on which $Class1_0^{(3)}$, $Buffer_0^{(1)}$ and $Buffer_0^{(3)}$ must synchronise.

4.3. Summary of case study

In a SAN, functional rates imply that some transitions rates of an automaton will depend on the state of one or several automata of the network. The use of such rates is a means to avoid explicitly modelling all parts of a system's behaviour. Modelling a system component implicitly in this way means that less automata are needed to model the complete system. In some cases, this may lead to a reduction in the size of the model. Moreover, benefits of this reduction can be appreciable when building/solving the underlying Markov chain.

Furthermore, the use of the functional rates within SAN appears to offer the modeller greater flexibility with respect to model construction. We saw this when we considered the two alternative representations of $queue_3$. In the second representation we were able to implicitly capture the local state of one component ($\mathcal{A}_{31}$) within another ($\mathcal{A}_{32}$) using the functional dependency. In the PEPA model a component representing only the local state of class 1 customers nevertheless needs access to the total number of customers. This can only be achieved by giving the complementary component, $Class2_0^{(3)}$ say, intended to represent class 2 customers, the ability to witness all state changes within the queue, i.e. the representation of $Class2_0^{(3)}$ is forced to become $Buffer_0^{(3)}$.

In the following, we investigate the introduction of the functional rates in PEPA and their impact on building the models not only in terms of flexibility, but also in terms of model size reduction and compact representation of the underlying Markov chain.

5. Functional dependencies in PEPA

In this section we define a modified version of PEPA in which functional dependencies are included and outline the benefits which can be gained by this addition. The modification, whilst fundamental, in fact stems from the modification of a single basic definition of PEPA: the range of values which may be used as activity rates in PEPA activities. Other modifications are largely syntactic.

The benefits can be summarised as follows:

- the modification results in greater modelling flexibility. This is demonstrated by considering again the previous example.
- a more elegant tensor representation of the generator of the underlying Markov process is possible when functional rates are available.

In the context of PEPA, a functional dependency may involve one or several components. In a functional dependency involving a single component, the rate value of an activity of the component depends on the current state of the component itself. This translates into the presence of several apparent rates for the activity in the component. Since each activity is represented explicitly in each local state it has always been possible to capture this form of dependency in PEPA. When this is expressed as a functional dependency, the rate value expressed as a function of the current component state is still a positive real number and can never be zero. While adding nothing new to the expressiveness of the language, it is this use of functional rates which leads to the compact tensor representation.

In contrast the ability to have an activity rate which is dependent on the local state of another component has not been possible (except in the special circumstance of cooperation). When this form of functional dependency is introduced into PEPA we may wish to allow the dependent rate to include the value zero, indicating that an activity is blocked by the local state of another component (as in the on/off source in the second example in Sect. 3.3). When the dependency is between two or more components it implies that either the activity to be performed by the first component and/or its rate value will be determined by the current state of the other component(s). The rate value may then be any non-negative real number including zero, particularly when the choice of the activity to be performed is done according to the state of another component.

In PEPA the set of activities $\mathcal{A}ct$ is defined as $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^+$ where $\mathbb{R}^+$ is the set of positive real numbers defined as follows:

$$\mathbb{R}^+ = \{r \mid r > 0;\ r \in \mathbb{R}\} \cup \{\top\}$$

The introduction of functional dependencies in PEPA therefore requires us to relax the constraint on the definition domain of an activity rate. Thus, the set of activities $\mathcal{A}ct$ is now defined as $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^*$ where $\mathbb{R}^*$ is the set of non-negative real numbers defined as follows:

$$\mathbb{R}^* = \{r \mid r \geqslant 0;\ r \in \mathbb{R}\} \cup \{\top\}$$

The syntax of sequential components is modified to allow an activity to defined in terms of an action type and an *expression* e, which can be either a rate, or a function, or a product of a rate and a function.

$$\begin{aligned} S &::= (\alpha, e).S \ \mid\ S + S \ \mid\ A \\ e &::= r \ \mid\ f \ \mid\ r \times f \end{aligned}$$

where $f : 2^{\mathcal{C}} \longrightarrow \mathbb{R}^*$ is a function from one or more components to the non-negative reals.

In Appendix A we show the operational semantics for PEPA, assuming that the duration of activities are specified via expressions. These rules are unchanged from the original rules when all expressions are assumed to be simple rates.

In the following, we show that, as in SAN, the functional dependencies in PEPA provide a flexibility in building the model. This is shown using the queueing network of the case study.

5.1. Modelling flexibility

Making use of functional rates, the system in Fig. 3 may now be modelled differently. The PEPA model is composed of four components: $Buffer_0^{(1)}$, $Buffer_0^{(2)}$, $Class1_0^{(3)}$ and $Class2_0^{(3)}$.

Let f be a function of x_1, the current derivative of $Class1_0^{(3)}$ and x_2 the current derivative of $Class2_0^{(3)}$ as follows:

$$f(x_1, x_2) = \begin{cases} 0 & \text{if } \ x_1 \equiv Class1_{C_3}^{(3)} \vee (x_1 \equiv Class1_{C_3-1}^{(3)} \wedge x_2 \equiv Class2_1^{(3)}) \vee \ldots \vee Class2_{C_3}^{(3)} \\ 1 & \text{otherwise} \end{cases}$$

where $\equiv$ denotes syntactic equivalence. Let $g(x_1, x_2) = 1 - f(x_1, x_2)$. To simplify, in the following we use f and g instead of $f(x_1, x_2)$ and $g(x_1, x_2)$. As in the previous PEPA model, $queue_1$ and $queue_2$ are modelled using $Buffer_0^{(1)}$ and $Buffer_0^{(2)}$ respectively as follows.

Component $Buffer_0^{(1)}$:

$$\begin{aligned} Buffer_0^{(1)} &\overset{def}{=} (in_1, \lambda_1).Buffer_1^{(1)} \\ Buffer_1^{(1)} &\overset{def}{=} (in_1, \lambda_1).Buffer_2^{(1)} + (service_1, \mu_1 \times f).Buffer_0^{(1)} + (blocking, \mu_1 \times g).Buffer_1^{(1)} \\ \vdots\quad & \quad\ \vdots \\ Buffer_{C_1}^{(1)} &\overset{def}{=} (loss_1, \lambda_1).Buffer_{C_1}^{(1)} + (service_1, \mu_1 \times f).Buffer_{C_1-1}^{(1)} + (blocking, \mu_1 \times g).Buffer_{C_1}^{(1)} \end{aligned}$$

Component $Buffer_0^{(2)}$:

$$\begin{aligned} Buffer_0^{(2)} &\overset{def}{=} (in_2, \lambda_2).Buffer_1^{(2)} \\ Buffer_1^{(2)} &\overset{def}{=} (in_2, \lambda_2).Buffer_2^{(2)} + (service_2, \mu_2 \times f).Buffer_0^{(2)} + (loss'_2, \mu_2 \times g).Buffer_0^{(2)} \\ \vdots\quad & \quad\ \vdots \\ Buffer_{C_2}^{(2)} &\overset{def}{=} (loss_2, \lambda_2).Buffer_{C_2}^{(2)} + (service_2, \mu_2 \times f).Buffer_{C_2-1}^{(2)} + (loss'_2, \mu_2 \times g).Buffer_{C_2-1}^{(2)} \end{aligned}$$

Opting for functional dependencies, $queue_3$ may now be represented by components $Class1_0^{(3)}$ and $Class2_0^{(3)}$. The first one models the total number of class 1 customers in $queue_3$ and the second the total number of class 2 customers in this queue. It is not necessary to explicitly represent information about the total number of customers in this queue. The information provided by components $Class1_0^{(3)}$ and $Class2_0^{(3)}$ is now sufficient.

Component $\boldsymbol{Class1_0^{(3)}}$:

$$\begin{array}{lcl}
Class1_0^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_1^{(3)} + (service_{32}, \top).Class1_0^{(3)} + (blocking, \top).Class1_0^{(3)} \\
Class1_1^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_2^{(3)} + (service_{31}, \mu_{31}).Class1_0^{(3)} + (blocking, \top).Class1_1^{(3)} \\
\vdots & & \vdots \\
Class1_i^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_{i+1}^{(3)} + (service_{31}, \mu_{31}).Class1_{i-1}^{(3)} + (blocking, \top).Class1_i^{(3)} \\
\vdots & & \vdots \\
Class1_{C_3}^{(3)} & \stackrel{def}{=} & (service_{31}, \mu_{31}).Class1_{C_3-1}^{(3)} + (blocking, \top).Class1_{C_3}^{(3)}
\end{array}$$

Component $\boldsymbol{Class2_0^{(3)}}$:

$$\begin{array}{lcl}
Class2_0^{(3)} & \stackrel{def}{=} & (service_2, \top).Class2_1^{(3)} + (loss'_2, \top).Class2_0^{(3)} \\
Class2_1^{(3)} & \stackrel{def}{=} & (service_2, \top).Class2_2^{(3)} + (service_{32}, \mu_{32} \times h).Class2_0^{(3)} + (loss'_2, \top).Class2_1^{(3)} \\
\vdots & & \vdots \\
Class2_j^{(3)} & \stackrel{def}{=} & (service_2, \top).Class2_{j+1}^{(3)} + (service_{32}, \mu_{32} \times h).Class2_{j-1}^{(3)} + (loss'_2, \top).Class2_j^{(3)} \\
\vdots & & \vdots \\
Class2_{C_3}^{(3)} & \stackrel{def}{=} & (service_{32}, \mu_{32} \times h).Class2_{C_3-1}^{(3)} + (loss'_2, \top).Class2_{C_3}^{(3)}
\end{array}$$

Here h is a function which depends on the current derivative of $Class1_0^{(3)}$, representing the class 1 customers in $queue_3$ as follows:

$$h(x_1) = \begin{cases} 1 & \text{if } \ x_1 \equiv Class1_0^{(3)} \\ 0 & \text{otherwise} \end{cases}$$

The complete system:

$$System \stackrel{def}{=} ((Buffer_0^{(1)} \underset{\mathcal{K}'}{\bowtie} Class1_0^{(3)}) \underset{\mathcal{M}'}{\bowtie} Class2_0^{(3)}) \underset{\mathcal{L}'}{\bowtie} Buffer_0^{(2)}$$

where $\mathcal{K}' = \{service_1, blocking\}$, $\mathcal{L}' = \{service_2, loss'_2\}$ and $\mathcal{M}' = \{service_{32}\}$.

Thus we now have the same modelling flexibility in PEPA as was previously apparent in SAN.

5.2. Reduction of model size

Besides the flexibility provided by the functional dependency, we may now be able to reduce the PEPA model size in some cases because parts of a system can be implicitly modelled. For example, consider a resource sharing system. In this system, N processors share M resources with $M \leqslant N$. We assume that the processors use and free the resources with different rates. Let λ_i and μ_i be the use and liberation rates respectively, characterising a processor i, $1 \leqslant i \leqslant N$.

To model the system, we may either use the functional dependencies or build a model in which explicit component interaction in the form of cooperation is used. To see the impact of the functional rates on model size, we consider both alternatives.

- **PEPA model** *without functions*

The behaviour of each processor i is modelled using component $Processor_0^{(i)}$. Moreover, we use component $NumberR_0$ to model the number of resources occupied by the processors. Two types of activities are considered: use_i and $free_i$. The former specifies that a processor i is using a resource whereas the latter specifies that it frees it. Formally these components are defined as follows.

Component $\boldsymbol{Processor_0^{(i)}}$, $1 \leqslant i \leqslant N$

$$\begin{array}{lcl}
Processor_0^{(i)} & \stackrel{def}{=} & (use_i, \lambda_i).Processor_1^{(i)} \\
Processor_1^{(i)} & \stackrel{def}{=} & (free_i, \mu_i).Processor_0^{(i)}
\end{array}$$

Component *NumberR*$_0$

$$\begin{array}{lcl}
NumberR_0 & \overset{\text{def}}{=} & \sum_{i=1}^{N}(use_i, \top).NumberR_1 \\
NumberR_1 & \overset{\text{def}}{=} & \sum_{i=1}^{N}(use_i, \top).NumberR_2 + \sum_{i=1}^{N}(free_i, \top).NumberR_0 \\
\dots & & \dots \\
NumberR_M & \overset{\text{def}}{=} & \sum_{i=1}^{N}(free_i, \top).NumberR_{M-1}
\end{array}$$

The complete system is then modelled as N components ($Processor_0^{(i)}$, $i = 1 \dots N$) which evolve independently, but which synchronise with component $NumberR_0$ as follows:

$$System \overset{\text{def}}{=} \left(Processor_0^{(1)} \mid\mid \dots \mid\mid Processor_0^{(N)}\right) \underset{K}{\bowtie} NumberR_0$$

where $K = \{use_1, \dots, use_N, free_1, \dots, free_N\}$

- **PEPA model** *with functions*
  In this model, where the functional rates are used, only components $Processor_0^{(i)}$, $1 \leqslant i \leqslant N$, are needed to capture the behaviour of the complete system. These are defined as follows:

$$\begin{array}{lcl}
Processor_0^{(i)} & \overset{\text{def}}{=} & (use_i, \lambda_i \times f(x_1, \dots, x_N)).Processor_1^{(i)} \\
Processor_1^{(i)} & \overset{\text{def}}{=} & (free_i, \mu_i).Processor_0^{(i)}
\end{array}$$

where x_i is a variable whose value is 1 if the current derivative of the ith processor is $Processor_1^{(i)}$, and 0 otherwise. The function f is then defined as follows:

$$f(x_1, \dots, x_N) = \begin{cases} 1 & \text{if } \sum_{j=1}^{N} x_j < M \quad x_j \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

The complete system is then modelled as N components ($Processor_0^{(i)}$, $i = 1 \dots N$) that evolve *apparently* in an independent manner. However this independence is only superficial because of the presence of function f.

$$System \overset{\text{def}}{=} Processor_0^{(1)} \mid\mid \dots \mid\mid Processor_0^{(N)}$$

This example illustrates how, in addition to increased modelling flexibility, the introduction of functional rates can reduce a model with respect to the number of components needed to express it. The model with functional rates has fewer components (N) than the one without ($N + 1$). The product state space of the former model has $2^N * (N + 1)$ states whereas the latter has 2^N states. However, both models have the same reachable state space; the underlying CTMC has same in both cases. Nevertheless, as we will see it later in Sect. 7.1, the reduction of the model's size remains an important gain.

In the remainder of the paper we will focus on another benefit of functional rates—that they facilitate a compact representation of the state space using tensor algebra. This can be regarded as an alternative abstraction of the model which has benefits for the quantitative analysis.

6. Tackling state space explosion

All state-based modelling formalisms, including SAN and PEPA, are prone to problems of state space explosion: the number of states generated in the underlying Markov process makes solution of the resulting matrix intractable.

There are several general approaches to overcoming the state space explosion problem; for example:

- The matrix representation may be decomposed so that the state space of the model, and its dynamics, are not represented by a single matrix but by a number of smaller matrices. Nevertheless the model is solved as a single entity.
- The model itself may be decomposed into a number of submodels, each of which is treated as a separate system generating a separate, smaller matrix. These matrices are solved separately, possibly in combination with a representation of the *aggregate model* which captures the interactions between submodels. In general this approach will give rise to approximate solution of the original model.

- The model may be replaced by another which is equivalent in some sense but which is more amenable to efficient solution, typically because its state space is smaller. This is termed *model simplification* and the accuracy of the approach depends on the model-model equivalence used to make the substitution.
- The mapping from model to matrix may be refined, based on an equivalence relation, so that one state at the Markov process level corresponds to an equivalence class of states in the model. This is termed *model aggregation*. This approach can result in a significant reduction in the size of the matrix, for example in models which exhibit symmetry due to repeated components. Whether this approach gives rise to exact or approximate solution with respect to the "complete" model will depend on the equivalence relation used.

In SAN a version of the first approach, based on tensor representation, has been incorporated into the formalism and this is presented in more detail in the following subsection. The use of a tensor representation of the underlying Markov process is not a state space reduction method. It is an alternative approach to state space explosion which handles the model solution in a decomposed form. Recently, a state space reduction technique has been developed for SANs [BBFP03]. This technique exploits aggregation based on replicated automata.

PEPA has developed equivalence relations for both model simplification and model aggregation. The interested reader is referred to [Hil94] for details. In Sect. 6.2 we present the bisimulation-based *strong equivalence* relation which is used as a basis for aggregation in PEPA.

6.1. Tensor representation

In the seminal paper [Pla85], Plateau proved that the generator matrix of the Markov process underlying a SAN model can be analytically represented using Kronecker algebra. Thus the generator matrix is stored as an expression involving the tensor sum and tensor product of smaller matrices (an introduction to tensor sum and tensor product can be found in Appendix C). The solution of the model can then be achieved via this tensor expression of submatrices—the complete matrix does not need to be generated.

The generator matrix Q is automatically derived from the SAN description, using the individual automata to generate the submatrices in the tensor expression (see [PFL88] for more details and proofs). It has been proved in [PFL88] that, if the states are in a lexicographic order, then the generator matrix Q_{san} of the Markov process associated with a continuous-time SAN is given by

$$Q_{san} = \bigoplus_{i=1}^{N} F_i + \sum_{e\in\varepsilon} \tau_e \left(\bigotimes_{i=1}^{N} S_{i,e} - \bigotimes_{i=1}^{N} \overline{S}_{i,e} \right)$$

where

- N is the total number of automata in the network and ε is the set of synchronisations.
- F_i is the transition matrix of automaton $\mathcal{A}_i$ without synchronisations.
- $S_{i,e}$ is the transition matrix of automaton $\mathcal{A}_i$ due to synchronisation e whose rate is τ_e.
- $\overline{S}_{i,e}$ is a matrix representing the normalisation associated with the synchronisation e on automaton $\mathcal{A}_i$.
- $\bigoplus$ and $\bigotimes$ denote tensor sum and product, respectively.

Unlike the local transition matrices F_i, the synchronising matrices $S_{i,e}$ are not generators (i.e. their rows do not sum to zero). The diagonal corrector matrices $\overline{S}_{i,e}$ have been introduced to normalise the synchronising matrices.

The tensorial representation of the generator matrix has an impact on both the memory requirements and the computation time of the matrix–vector multiplication. The space efficiency of this representation is obtained at the expense of an increased computation time. Therefore, an efficient vector–tensor product multiplication algorithm has been developed, *vector–descriptor multiplication* [FPA98] to be used when solving the stationary distribution. This algorithm is the basic step in iterative methods such as the Power method and Generalized Minimum Residual (GMRES) method [SAP95]. In [Day98], iterative methods based on splittings, such as Jacobi and Gauss–Seidel, are proved to be better than the Power method.

6.2. Aggregation based on strong equivalence

Strong equivalence[2] is a bisimulation-based equivalence relation for PEPA and when it is applied to the states within a single model it induces a partition over the state space of the model. When aggregation based on this partition is carried out, a matrix representation for the model is constructed so that there is one row (column) of the matrix for each equivalence class of states in the process algebra model.

In PEPA, two derivatives are strongly equivalent if there is an equivalence relation between them such that, for any action type α, the *total conditional transition rates* from those derivatives to any equivalence class are the same. The conditional transition rate between two derivatives P and P', via a given action type α, denoted $q(P, P', \alpha)$, is the sum of the activity rates associated with transitions between P and P' in the derivation graph which are labelled by α. The total conditional transition rate from a derivative P to a set of derivatives S via a given action type α, denoted $q[P, S, \alpha]$ is defined to be

$$q[P, S, \alpha] = \sum_{P' \in S} q(P, P', \alpha)$$

Definition 6.1 An equivalence relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ is a *strong equivalence* if whenever $(P, Q) \in \mathcal{R}$ then for all $\alpha \in \mathcal{A}$ and for all $S \in \mathcal{C}/R$

$$q[P, S, \alpha] = q[Q, S, \alpha]$$

In general, if the states of a Markov process are partitioned and a new stochastic process constructed with one state for each partition, the resulting process does not have the Markov, or memoryless, property. For this to be the case the partition must satisfy a condition on transition rates which is termed *lumpability*. In [Hil94] it is proved that a partition induced by strong equivalence is always lumpable and consequently results calculated via this aggregation are exact. Furthermore, because strong equivalence is a congruence, the technique can be applied compositionally, replacing cooperating components by strongly equivalent (lumped) ones—it has been proved that each component is equivalent to its lumped version and that lumped components can be substituted one at a time. Note that this compositional application of the aggregation procedure does not necessarily give the maximal lumping but for very large models it avoids the construction of the complete state space before aggregation can begin (which would otherwise be the case).

7. Exploiting functional rates in PEPA

In this section we consider how we can exploit the introduction of functional dependencies in PEPA with respect to techniques for tackling state space explosion introduced in the previous section.

7.1. The tensor representation for PEPA

In [HK01], we have shown that PEPA models can be represented analytically using Kronecker algebra and solved without constructing the complete generator matrix. Moreover, the translation from the model to the compact representation is automatic.

Firstly, we capture local transitions for each constituent component in an appropriate generator matrix. Then we represent each type of interaction by the tensor product of matrices capturing each component's capacity to participate in this shared activity. In order to ensure that the rate of the shared activity is represented correctly we need to exercise a little care in constructing this set of tensor products. Finally, and analogously to the SAN case, we need to introduce diagonal corrector matrices which normalise the cooperation matrices.

In PEPA without functional dependencies the tensor product capturing component interactions needs to distinguish each apparent rate of each action type in each component, in order to correctly calculate the rate of a shared activity. Furthermore, we need to introduce a new tensorial operator, the *minimum tensor product*, to capture the interaction between two components. This operator, denoted $\bigotimes_{min}$, is based on the Kronecker product and operates on (scalar, matrix) pairs.

[2] Sometimes termed *Markovian bisimulation.*

Definition 7.1 *Consider two scalars s_1 and s_2, and two matrices M_1 and M_2. The minimum tensor product of (s_1, M_1) and (s_2, M_2) is defined as follows:*

$$(s_1, M_1) \bigotimes_{min} (s_2, M_2) = \min(s_1, s_2)\,(M_1 \otimes M_2)$$

This form of product is used to capture shared activities and so the scalar represents the apparent rate of an activity in one component, with the corresponding matrix capturing the associated possible state transitions as a probability transition matrix. If an action type is passive within a component its apparent rate is $\top$ and for all real values s, $min(s, \top) = s$.

The generator matrix Q'_{pepa} of the Markov process associated with a PEPA model is given by

$$Q'_{pepa} = \bigoplus_{i=1}^{N} P'_i + \sum_{\alpha \in \mathcal{Z}} \bigotimes_{min}{}_{i=1}^{N} \sum_{r_{\alpha,j} \in R_i(\alpha)} (r_{\alpha,j}, P'_{i,\alpha,r_{\alpha,j}}) - \sum_{\alpha \in \mathcal{Z}} \bigotimes_{min}{}_{i=1}^{N} \sum_{r_{\alpha,j} \in R_i(\alpha)} (r_{\alpha,j}, \overline{P'}_{i,\alpha,r_{\alpha,j}}) \tag{7.1}$$

where

- $R_i(\alpha)$ is the set of apparent rates for α enabled by derivatives of C_i.
- $P'_{i,\alpha,r_{\alpha,j}}$ is the probability transition matrix of component C_i due to activities of type α when the apparent rate of α is $r_{\alpha,j}$.
- $\overline{P'}_{i,\alpha,r_{\alpha,j}}$ is a matrix representing the normalisation associated with the shared activity α at apparent rate $r_{\alpha,j}$ in component C_i.

However, when we have functional dependencies within PEPA we can capture the different apparent rates that a component may express with respect to an action type using a functional dependency on the state of that component. This allows the tensor representation to be more direct and similar to the one obtained by Plateau for the SANs. If Q_{pepa} is the generator matrix of the Markov process associated with a PEPA model, its tensor expression is defined as follows:[3]

$$Q_{pepa} = \bigoplus_{i=1}^{N} P_i + \sum_{\alpha \in \mathcal{Z}} r_\alpha \left(\bigotimes_{i=1}^{N} P_{i,\alpha} - \bigotimes_{i=1}^{N} \overline{P}_{i,\alpha} \right) \tag{7.2}$$

where

- N is the total number of components in the PEPA model and $\mathcal{Z}$ is the set of cooperating action types.
- P_i is the transition matrix of component C_i relating solely to its individual activities.
- r_α is the minimum of the functional rates of action type α over all components C_i, $i = 1 \ldots N$: $r_\alpha = \min_{i=1\ldots N} (r_\alpha(C_i))$.
- $P_{i,\alpha}$ is the probability transition matrix of component C_i due to activity of type α. Its elements values are either 1 or 0.
- $\overline{P}_{i,\alpha}$ is a matrix representing the normalisation associated with the shared activity α in component C_i.

As previously, the local transition matrices P_i, the cooperation matrices $P_{i,\alpha}$ are not generators. So we need to introduce diagonal corrector matrices $\overline{P}_{i,\alpha}$ to normalise the cooperation matrices.

Note that in order to achieve the greater simplicity of this expression, in [HK01] functional rates were introduced into PEPA purely to facilitate the tensor representation. They were used only in this underlying representation; they were not incorporated at the modelling level, i.e. in the syntax of the language as in Sect. 5.

In models with a tensor representation, the "*size*" of the state space is open to several interpretations:

1. the physical space needed to store the model using the tensor representation;
2. the size of the state space of the cartesian product of the model components;
3. the size of the space of the reachable states.

[3] There is an implicit assumption that an action type uniquely defines a synchronisation event at the transition system level. This will not generally be the case without restrictions on the use of types within cooperation sets. These are not terribly strong restrictions and can be achieved by pre-processing (see [HK01] for full details).

From previous work on SAN, we know that the use of functions has a positive effect on both the size of the tensor representation and the size of the product state space. In particular if we remove a function it is generally necessary to introduce an additional component. If the new component has two or more states then we increase both spaces (1) and (2). However this should not change the reachable state space (3). To do so would fundamentally change the model considered but note that in the tensor representations for SAN the cartesian product space is represented, not the reachable state space.

7.2. Functional strong equivalence and aggregation

As explained in Sect. 6.2 model aggregation in PEPA is governed by the equivalence relation strong equivalence. Thus in order to consider the impact of functional rates on the aggregation technique we must first consider the impact of functional rates on strong equivalence.

Firstly, we must extend the definition of conditional transition rate to include the possibility that the transition rate concerned may be a function.

Definition 7.2 *The* conditional transition rate *between two derivatives C_i and C_j, via a given action type α, denoted $q(C_i, C_j, \alpha)$, is defined to be the sum of the constant and the functional activity rates associated with transitions between C_i and C_j in the derivation graph which are labelled by α.*

Note that the evaluation of a function is unequivocal because we are considering the transition rates from a particular derivative. Each derivative corresponds to a particular set of local states for each component, thus determining the appropriate value of the function. As a consequence the conditional transition rate and total conditional transition rate from any derivative has a literal value even if it is expressed via a function.

Properties of the Strong Equivalence: By a similar argument—that in any given state the functions can be evaluated unequivocally—we are able to establish that functional strong equivalence $\cong$ is a congruence for PEPA; the relation is preserved by the combinators and the recursive definitions.

Proposition 7.1 If $P \cong Q$ then

1. $(a, e).P \cong (a, e).Q$
2. $P + S \cong Q + S$
3. $P \underset{L}{\bowtie} S \cong Q \underset{L}{\bowtie} S$
4. $P/L \cong Q/L$

Proof. The proof is omitted as it is similar to the one developed in [Hil94]. □

Proposition 7.2 *Let $\tilde{E}$ and $\tilde{F}$ be two indexed sets of components which contain the indexed set of variables $\tilde{X}$ at most. Let $\tilde{A} \overset{\text{def}}{=} \tilde{E}\{\tilde{A}/\tilde{X}\}$ and $\tilde{B} \overset{\text{def}}{=} \tilde{F}\{\tilde{B}/\tilde{X}\}$. If $\tilde{E} \cong \tilde{F}$ then $\tilde{A} \cong \tilde{B}$.*

Proof. Again, the proof is similar to the one developed in [Hil94] and is omitted here. □
The consequence of these results is that the aggregation technique based on strong equivalence which has been developed for PEPA models can equally be applied to PEPA models with functional rates.

In the following section we consider a PEPA case study in which we include functional rates and demonstrate the construction of the tensor representation and the application of the aggregation technique, and discuss the interplay between these two techniques.

8. Case study 2: hierarchical cellular network

This case study is inspired by the work presented in [FKV02]. Fourneau et al. investigate the performance of a hierarchical cellular network using PEPA. They do not use functional dependencies in their model. Here we show how the network may be modelled using functional rates to avoid several apparent rates for an action type in a component; we demonstrate how the model should be pre-processed in order to build the PEPA descriptor of the model using equation 7.2; and finally, we apply the aggregation technique as in [FKV02].

We consider a hierarchical cellular network based on the Manhattan model of a city. This model consists of square blocks, representing buildings, with streets in between them. Thus, the reuse pattern is composed of a

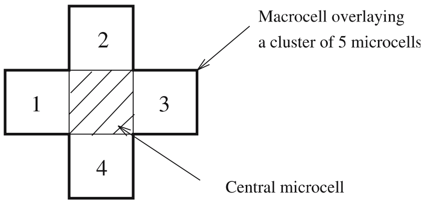


Fig. 6. The reuse pattern

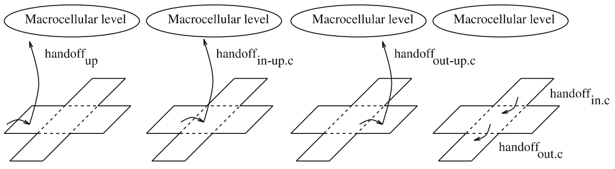


Fig. 7. The handoff activities graph

macro-cell overlying a cluster of $N = 5$ micro-cells: a central micro-cell surrounded by four peripheral micro-cells (see Fig. 6).

In this study we assume that the macro-cell and all the micro-cells have the same capacity, $c_i = 3$, $i = 0 \dots 5$. We consider two types of customers inside the cluster, the new calls and the handover requests (ongoing calls). External arrivals to the cluster consist of the handover requests coming from other clusters and the new calls initiated in that cluster. The handover requests coming from other clusters may occur only in the macro-cell or the peripheral micro-cells. They may never occur in the central micro-cell.

We consider that the new calls can be assigned only to the macro-cell level. Moreover, we assume that a request, either a new call or a handover request, initiated at the micro-cell level is served in its originating micro-cell if a channel is available. Otherwise, according to the overflow strategy, the request is overflowed to the upper level. In the case where all channels are busy at both levels, the request is dropped (handover) or blocked (new call).

8.1. The PEPA model

In the model, the external arrival process is represented by an *in* activity by the cells. The arrival rate is assumed to be λ_1 in the macro-cell, λ_2 in the peripheral micro-cells and λ_3 in the central micro-cell.

Because of the different types of cells (macro-cell, peripheral micro-cell, central micro-cell) and the topology of the network, we make a distinction between the handover requests generated inside the cluster. This distinction is based on the cell type the call originated from and the cell type the call has to be transferred to. Thus we have the following activities:

- activity $handoff_{up}$ represents the process which transfers a call from a micro-cell to the macro-cell. This call (a new call or a handover call) comes from outside the cluster to the micro-cell and because all channels in the micro-cell are busy, it has to be transferred to the macro-cell. The rate of this activity is the external arrival rate to the micro-cell;
- activity $handoff_{in.c}$ represents the transfer of an ongoing call from one of the peripheral micro-cells to the central micro-cell;
- in contrast, activity $handoff_{out.c}$ represents the transfer of an ongoing call from the central micro-cell to one of the four peripheral micro-cells;
- activity $handoff_{in-up.c}$ models the process in which an ongoing call coming from a peripheral micro-cell and entering the central micro-cell, is then transferred to the macro cell because all channels of the central micro-cell are busy;
- activity $handoff_{out-up.c}$ models the arrival of an ongoing call from the central micro-cell to a peripheral micro-cell when all channels of this cell are busy, and which is overflowed to the macro cell.

As the process behind the four last *handoff* activities is the same, the rate of these activities is also the same and it is denoted δ (representing the mean dwell-time in a micro-cell). In all cells, the service process is represented by activity *service*. As the service rate in each cell is assumed to be μ, when there are i, $1 \leqslant i \leqslant c_k$, customers in a cell, it will engage in a *service* activity at rate $i\,\mu$.

Now, let us give the behaviour details of the different components of the system.

Component *macro* When the channels of the macro-cell are not all busy, if external handover calls arrive at rate λ_1, then *macro* will engage in an *in* activity at rate λ_1. If handover calls arrive from the micro-cells, then *macro* will engage in either a $handoff_{up}$ activity, or a $handoff_{in-up.c}$ activity, or a $handoff_{out-up.c}$ activity. In all cases, the rate of the activity is unspecified ($\top$) in component *macro* because these activities synchronise with activities of micro-cells generating the handover calls. The rate will be determined by the micro-cell which generates the handover calls.

When all channels are busy, if handover calls arrive from the micro-cells, *macro* will engage in the *handoff* activity but the ongoing call will be dropped and thus lost. Similarly, if external handover calls arrive when all channels are busy, *macro* will engage in the *in* activity but the call will be blocked and lost.
If $macro_i$ describes the component behaviour when there are i customers in the macro-cell, this behaviour is as follows:

$$\begin{aligned}
macro_0 &\overset{def}{=} (in, \lambda_1).macro_1 + (handoff_{up}, \top).macro_1 + (handoff_{in-up.c}, \top).macro_1 \\
&\quad + (handoff_{out-up.c}, \top).macro_1 \\[1ex]
macro_1 &\overset{def}{=} (in, \lambda_1).macro_2 + (service, \mu).macro_0 + (handoff_{up}, \top).macro_2 \\
&\quad + (handoff_{in-up.c}, \top).macro_2 + (handoff_{out-up.c}, \top).macro_2 \\[1ex]
macro_2 &\overset{def}{=} (in, \lambda_1).macro_3 + (service, 2\,\mu).macro_1 + (handoff_{up}, \top).macro_3 \\
&\quad + (handoff_{in-up.c}, \top).macro_3 + (handoff_{out-up.c}, \top).macro_3 \\[1ex]
macro_3 &\overset{def}{=} (in, \lambda_1).macro_3 + (service, 3\,\mu).macro_2 + (handoff_{up}, \top).macro_3 \\
&\quad + (handoff_{out-up.c}, \top).macro_3 + (handoff_{in-up.c}, \top).macro_3
\end{aligned}$$

Component $micro_j : 1 \leqslant j \leqslant 4$ As for the macro component, external arrivals to peripheral micro cells (new calls or handover calls) are modelled by an *in* activity at rate λ_2.

When handover requests from the central micro-cell arrive, if channels are not all busy, $micro_j$ will engage in a $handoff_{out.c}$ activity at an unspecified rate. If all channels are busy, $micro_j$ will then engage in a $handoff_{out-up.c}$ activity. Component $micro_j$ may also engage in a $handoff_{up}$ activity at rate λ_2, if external calls arrive when all channels are busy.

A peripheral micro-cell may generate handover calls which will ask for channels from the central micro-cell. $micro_j$ will engage in a $handoff_{in.c}$ activity at rate $i\delta$ where i is the number of customers in this cell.
If $micro_{ji}$ denotes the component behaviour when there are i customers in the peripheral micro-cell j, this behaviour is as follows:

$$\begin{aligned}
micro_{j0} &\overset{def}{=} (in, \lambda_2).micro_{j1} + (handoff_{out.c}, \top).micro_{j1} \\[1ex]
micro_{j1} &\overset{def}{=} (in, \lambda_2).micro_{j2} + (handoff_{out.c}, \top).micro_{j2} + (service, \mu).micro_{j0} \\
&\quad + (handoff_{in.c}, g \times \delta).micro_{j0} + (handoff_{in-up.c}, g \times \delta).micro_{j0} \\[1ex]
micro_{j2} &\overset{def}{=} (in, \lambda_2).micro_{j3} + (handoff_{out.c}, \top).micro_{j3} + (service, 2\,\mu).micro_{j1} \\
&\quad + (handoff_{in.c}, g \times \delta).micro_{j1} + (handoff_{in-up.c}, g \times \delta).micro_{j1} \\[1ex]
micro_{j3} &\overset{def}{=} (service, 3\,\mu).micro_{j2} + (handoff_{in-up.c}, g \times \delta).micro_{j2} + (handoff_{in.c}, g \times \delta).micro_{j2} \\
&\quad + (handoff_{up}, \lambda_2).micro_{j3} + (handoff_{out-up.c}, \top).micro_{j3}
\end{aligned}$$

Here g is a function whose value depends on the current derivative of the same peripheral micro-cell, $micro_{ji}$: $g(micro_{ji}) = i$, $1 \leqslant i \leqslant 3$.

Component $micro_C$ The $micro_C$ component differs from the other micro-cells only by the fact that it can receive handover calls generated by the four peripheral cells. So, it engages in the same type of activities and the same reasoning is used to describe its behaviour.

Let $micro_{Ci}$ denote the component behaviour when there are i customers in the central micro-cell. And let f be a function whose value depends on the current derivative of this micro-cell, $f(micro_{Ci}) = i,\ 1 \leqslant i \leqslant 3$. The behaviour of the central micro-cell is as follows:

$$\begin{aligned}
micro_{C0} &\overset{def}{=} (in, \lambda_3).micro_{C1} + (handoff_{in.c}, \top).micro_{C1} \\[1em]
micro_{C1} &\overset{def}{=} (in, \lambda_3).micro_{C2} + (service, \mu).micro_{C0} + (handoff_{in.c}, \top).micro_{C2} \\
&\quad + (handoff_{out-up.c}, f \times \delta).micro_{C0} + (handoff_{out.c}, f \times \delta).micro_{C0} \\[1em]
micro_{C2} &\overset{def}{=} (in, \lambda_3).micro_{C3} + (service, 2\,\mu).micro_{C1} + (handoff_{in.c}, \top).micro_{C3} \\
&\quad + (handoff_{out-up.c}, f \times \delta).micro_{C1} + (handoff_{out.c}, f \times \delta).micro_{C1} \\[1em]
micro_{C3} &\overset{def}{=} (service, 3\,\mu).micro_{C2} + (handoff_{up}, \lambda_3).micro_{C3} + (handoff_{out.c}, f \times \delta).micro_{C2} \\
&\quad + (handoff_{in-up.c}, \top).micro_{C3} + (handoff_{out-up.c}, f \times \delta).micro_{C2}
\end{aligned}$$

The system is formed by the cooperation of *macro* and the different micro-cells. Since the four peripheral micro-cells proceed independently, and cooperate with the central micro-cell, the system is defined as follows:

$$System \overset{def}{=} \Big((micro_{10} \mid\mid micro_{20} \mid\mid micro_{30} \mid\mid micro_{40}) \underset{\mathcal{L}}{\bowtie} micro_{C0}\Big) \underset{\mathcal{K}}{\bowtie} macro_0$$

where $\mathcal{L} = \{handoff_{in.c},\ handoff_{out.c}\}$ is the set of activities on which the central micro-cell and the peripheral micro-cells must synchronise. The set $\mathcal{K} = \{handoff_{up},\ handoff_{in-up.c},\ handoff_{out-up.c}\}$ contains the activities on which the macro-cell and the micro-cells must synchronise.

This model has 4096 states and 81920 transitions.

8.2. The model descriptor

In this section, we show how to build, according to the generator expression 7.2, the different matrices corresponding to the hierarchical cellular network model.

The local transition matrices P_i:

- Component $macro_0$

$$P_0 = \begin{pmatrix} -\lambda_1 & \lambda_1 & 0 & 0 \\ \mu & -(\lambda_1 + \mu) & \lambda_1 & 0 \\ 0 & 2\,\mu & -(\lambda_1 + 2\,\mu) & \lambda_1 \\ 0 & 0 & 3\,\mu & -3\,\mu \end{pmatrix}$$

- Components $micro_{j0},\ j = 1 \dots 4$

$$P_j = \begin{pmatrix} -\lambda_2 & \lambda_2 & 0 & 0 \\ \mu & -(\lambda_2 + \mu) & \lambda_2 & 0 \\ 0 & 2\,\mu & -(\lambda_2 + 2\,\mu) & \lambda_2 \\ 0 & 0 & 3\,\mu & -3\,\mu \end{pmatrix}$$

- Component $micro_{C0}$

$$P_5 = \begin{pmatrix} -\lambda_3 & \lambda_3 & 0 & 0 \\ \mu & -(\lambda_3 + \mu) & \lambda_3 & 0 \\ 0 & 2\,\mu & -(\lambda_3 + 2\,\mu) & \lambda_3 \\ 0 & 0 & 3\,\mu & -3\,\mu \end{pmatrix}$$

The cooperation matrices $P_{i,\alpha}$: Before building the cooperation matrices, we need to rename the cooperation action types in which $micro_j$ is involved. This model pre-processing is due to the fact that only one peripheral micro-cell may cooperate with the macro-cell or the central micro-cell at once. Therefore the set of cooperating action types $\mathcal{Z}$, which initially is defined as $\mathcal{Z} = \mathcal{L} \cup \mathcal{K}$, becomes as follows:

$$\mathcal{Z} = \{handoff_{in.c_j},\ handoff_{out.c_j},\ handoff_{up},\ handoff_{up_j},\ handoff_{in-up.c_j},\ handoff_{out-up.c_j}\}$$

where $j = 1, \ldots, 4$. Now, we build the cooperation matrices where we define k as $k = 1, \ldots, 4$.

- **Action type** $\alpha_1 = handoff_{in.c_j}$. For each value of $j,\ j = 1 \ldots 4$, $r_{\alpha_1} = g \times \delta$ and

$$P_{0,\alpha_1} = P_{k,\alpha_1} = I_d,\quad k \neq j,\quad P_{j,\alpha_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix},\quad P_{5,\alpha_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

where I_d is the Identity matrix.

- **Action type** $\alpha_2 = handoff_{out.c_j}$ with $r_{\alpha_2} = f \times \delta$. For each value of $j,\ j = 1 \ldots 4$, we have:

$$P_{0,\alpha_2} = P_{k,\alpha_2} = I_d,\quad k \neq j,\quad P_{j,\alpha_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix},\quad P_{5,\alpha_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- **Action type** $\alpha_3 = handoff_{up}$ with $r_{\alpha_3} = \lambda_3$

$$P_{0,\alpha_3} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},\quad P_{j,\alpha_3} = I_d,\quad j = 1, \ldots, 4,\quad P_{5,\alpha_3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **Action type** $\alpha_4 = handoff_{up_j}$ with $r_{\alpha_4} = \lambda_2$. For each value of $j, j = 1, \ldots, 4$, we have:

$$P_{0,\alpha_4} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},\quad P_{j,\alpha_4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\quad P_{k,\alpha_4} = P_{5,\alpha_4} = I_d,\quad k \neq j$$

- **Action type** $\alpha_5 = handoff_{in-up.c_j}$. For each value of $j, j = 1, \ldots, 4$, $r_{\alpha_5} = g \times \delta$ and

$$P_{0,\alpha_5} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},\quad P_{j,\alpha_5} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P_{k,\alpha_5} = I_d,\quad k \neq j,\quad P_{5,\alpha_5} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **Action type** $\alpha_6 = handoff_{out-up.c_j}$ with $r_{\alpha_6} = f \times \delta$. For each value of $j, j = 1, \ldots, 4$, we have:

$$P_{0,\alpha_6} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},\quad P_{j,\alpha_6} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P_{k,\alpha_6} = I_d, \quad k \neq j, \quad P_{5,\alpha_6} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Using these matrices, we can then build the generator matrix as follows:

$$\begin{aligned} Q = {} & \bigoplus_{i=0}^{5} P_i + \sum_{j=1}^{4} g\delta \left(\bigotimes_{i=0}^{5} P_{i,\alpha_1} - \bigotimes_{i=0}^{5} \overline{P}_{i,\alpha_1} \right)_j \\ & + \sum_{j=1}^{4} f\delta \left(\bigotimes_{i=0}^{5} P_{i,\alpha_2} - \bigotimes_{i=0}^{5} \overline{P}_{i,\alpha_2} \right)_j + \textstyle\sum_{j=1}^{4} \lambda_2 \left(\bigotimes_{i=0}^{5} P_{i,\alpha_4} - \bigotimes_{i=0}^{5} \overline{P}_{i,\alpha_4} \right)_j \\ & + \lambda_3 \left(\bigotimes_{i=0}^{5} P_{i,\alpha_3} - \bigotimes_{i=0}^{5} \overline{P}_{i,\alpha_3} \right) + \textstyle\sum_{j=1}^{4} g\delta \left(\bigotimes_{i=0}^{5} P_{i,\alpha_5} - \bigotimes_{i=0}^{5} \overline{P}_{i,\alpha_5} \right)_j \\ & + \textstyle\sum_{j=1}^{4} f\delta \left(\bigotimes_{i=0}^{5} P_{i,\alpha_6} - \bigotimes_{i=0}^{5} \overline{P}_{i,\alpha_6} \right)_j \end{aligned}$$

8.3. The model aggregation

In this section we show how to apply the aggregation technique to this model. The top-level components in this model are the parallel composition of the peripheral micro-cells, the central micro-cell and the macro-cell. The atomic components are the individual cells. The components which exhibit the same behaviour are the four peripheral micro-cells. Consider the component representing these micro-cells in the system:

$$micro = micro_{10} \parallel micro_{20} \parallel micro_{30} \parallel micro_{40}$$

We illustrate the approach applying the technique to the component $micro_{10} \parallel micro_{20}$. Applying the technique subsequently to $micro_{30} \parallel micro_{40}$ we obtain two strongly equivalent, lumped components, on which we again apply the aggregation technique to obtain the final strongly equivalent, lumped component of the original component.

Consider the component $micro_{10} \parallel micro_{20}$. The derivation set, denoted $ds(micro_{10} \parallel micro_{20})$, of this component is as follows:

$$\begin{aligned} ds(micro_{10} \parallel micro_{20}) = \{ \; & micro_{10} \,||\, micro_{20},\; micro_{1k} \,||\, micro_{20},\; micro_{10} \,||\, micro_{2k},\; micro_{1k} \,||\, micro_{2l}, \\ & micro_{13} \,||\, micro_{20},\; micro_{10} \,||\, micro_{23},\; micro_{13} \,||\, micro_{2l},\; micro_{1l} \,||\, micro_{23}, \\ & micro_{13} \,||\, micro_{23},\quad 1 \leqslant k, l \leqslant 2\} \end{aligned}$$

Partitioning the derivative set of our component by strong equivalence results in the following set of equivalence classes:

$$\begin{aligned} ds(micro_{10} \,||\, micro_{20})/_{\cong} = \{ \; & [micro_{10} \,||\, micro_{20}],\; [micro_{1k} \,||\, micro_{20},\; micro_{10} \,||\, micro_{2k}],\; [micro_{13} \,||\, micro_{23}] \\ & [micro_{1k} \,||\, micro_{2l}, micro_{1l} \,||\, micro_{2k}],\; [micro_{13} \,||\, micro_{20},\; micro_{10} \,||\, micro_{23}], \\ & [micro_{13} \,||\, micro_{2l},\; micro_{1l} \,||\, micro_{23}],\quad 1 \leqslant k, l \leqslant 2\} \end{aligned}$$

We now form the lumped component, denoted mm. We associate one derivative of mm with each node of the lumped derivation graph as follows:

$$\begin{array}{llll} mm_{00} \longleftrightarrow [micro_{10} \,||\, micro_{20}] & & mm_{k0} \longleftrightarrow [micro_{1k} \,||\, micro_{20},\; micro_{10} \,||\, micro_{2k}] \\ mm_{kl} \longleftrightarrow [micro_{1k} \,||\, micro_{2l},\; micro_{1l} \,||\, micro_{2k}] & & mm_{30} \longleftrightarrow [micro_{13} \,||\, micro_{20},\; micro_{10} \,||\, micro_{23}] \\ mm_{3l} \longleftrightarrow [micro_{13} \,||\, micro_{2l},\; micro_{1l} \,||\, micro_{23}] & & mm_{33} \longleftrightarrow [micro_{13} \,||\, micro_{23}] \end{array}$$

The derivative component mm_{kl} models the behaviour of the lumped component mm when there are k customers in one of the two micro-cells and l in the other. Using the lumped activity sets, we can define the behaviour of these lumped components:

For $1 \leqslant k, l \leqslant 2$

$$\begin{array}{lcl}
mm_{00} & \overset{\text{def}}{=} & (in, \lambda_2).mm_{10} + (handoff_{out.c}, \top).mm_{10} \\
mm_{k0} & \overset{\text{def}}{=} & (in, \lambda_2).mm_{k1} + (in, \lambda_2).mm_{(k+1)0} + (service, k \times \mu).mm_{(k-1)0} \\
 & & + (handoff_{in.c}, h \times \delta).mm_{(k-1)0} + (handoff_{in-up.c}, h \times \delta).mm_{(k-1)0} \\
 & & + (handoff_{out.c}, \top).mm_{k1} + (handoff_{out.c}, \top).mm_{(k+1)0} \\
mm_{kl} & \overset{\text{def}}{=} & (in, \lambda_2).mm_{k(l+1)} + (in, \lambda_2).mm_{(k+1)l} \\
 & & + (service, k \times \mu).mm_{(k-1)l} + (service, l \times \mu).mm_{k(l-1)} \\
 & & + (handoff_{in.c}, h \times \delta).mm_{k(l-1)} + (handoff_{in.c}, h \times \delta).mm_{(k-1)l} \\
 & & + (handoff_{out.c}, \top).mm_{k(l+1)} + (handoff_{out.c}, \top).mm_{(k+1)l} \\
 & & + (handoff_{in-up.c}, h \times \delta).mm_{k(l-1)} + (handoff_{in-up.c}, h \times \delta).mm_{(k-1)l}
\end{array}$$

For $k = 3$ **and** $1 \leqslant l \leqslant 2$

$$\begin{array}{lcl}
mm_{30} & \overset{\text{def}}{=} & (in, \lambda_2).mm_{31} + (service, 3 \times \mu).mm_{20} + (handoff_{out.c}, \top).mm_{31} \\
 & & + (handoff_{in.c}, h \times \delta).mm_{20} + (handoff_{up}, \lambda_2).mm_{30} \\
 & & + (handoff_{in-up.c}, h \times \delta).mm_{20} + (handoff_{out-up.c}, \top).mm_{30} \\
mm_{3l} & \overset{\text{def}}{=} & (in, \lambda_2).mm_{3(l+1)} + (service, l \times \mu).mm_{3(l-1)} + (service, 3 \times \mu).mm_{2l} \\
 & & + (handoff_{in.c}, h \times \delta).mm_{3(l-1)} + (handoff_{in.c}, h \times \delta).mm_{2l} \\
 & & + (handoff_{in-up.c}, h \times \delta).mm_{3(l-1)} + (handoff_{in-up.c}, h \times \delta).mm_{2l} \\
 & & + (handoff_{out-up.c}, \top).mm_{3l} + (handoff_{out.c}, \top).mm_{3(l+1)} + (handoff_{up}, \lambda_2).mm_{3l} \\
mm_{33} & \overset{\text{def}}{=} & (service, 2 \times 3 \times \mu).mm_{32} + (handoff_{in.c}, h \times \delta).mm_{32} \\
 & & + (handoff_{up}, 2 \times \lambda_2).mm_{33} + (handoff_{out-up.c}, \top).mm_{33} \\
 & & + (handoff_{in-up.c}, h \times \delta).micro_{32}
\end{array}$$

This component, mm_{00}, now replaces component $micro_{10} \; || \; micro_{20}$ in the complete model. Function h depends on both the current state of the component and its next state. We define this function as follows:

$$h(mm_{kl}, mm_{k'l'}) = \begin{cases} k & \text{if } k' = k - 1 \\ l & \text{if } l' = l - 1 \\ 2 \times 3 & \text{if } k = l = 3 \end{cases}$$

where mm_{kl} and $mm_{k'l'}$ are respectively the current state and the next state of the component.

We can proceed similarly for the component $micro_{30} \parallel micro_{40}$ leading to a lumped component mm^* analogous to component mm. Component, mm^*_{00}, now replaces component $micro_{30} \; || \; micro_{40}$ in the complete model and the system is modelled as follows:

$$System \overset{\text{def}}{=} \left((mm_{00} \parallel mm^*_{00}) \underset{\mathcal{L}}{\bowtie} micro_{C0} \right) \underset{\mathcal{K}}{\bowtie} macro_0$$

The last step of the reduction of the model is to consider component $mm_{00} \parallel mm^*_{00}$ and replace it by an aggregate which we denote $micro_{0000}$. The complete aggregation technique applied to this model (without functional rates) may be found in detail in [FKV00].

After applying the aggregation technique, we obtain a model composed of only three components: $macro_0$, $micro_{C0}$ and $micro_{0000}$. A derivative $micro_{ijkl}$, $0 \leqslant i, j, k, l, \leqslant 3$ models the behaviour of the lumped component $micro$ when there are i customers in one of the microcells, j customers in another microcell, k customers in another and l in the last one. All combinations of i, j, k and l are considered.

The new system component is the following:

$$System^* \overset{\text{def}}{=} \left(micro_{0000} \underset{\mathcal{L}}{\bowtie} micro_{C0} \right) \underset{\mathcal{K}}{\bowtie} macro_0$$

This model has now 2560 states and 46016 transitions instead of 4096 states and 81920 transitions.

8.4. The aggregated model descriptor

Applying the aggregation technique to our model component *System* reduces the number of components by half; the resulting model $System^*$ has only three components: $macro_0$, $micro_{0000}$ and $micro_{C0}$. In the following we will refer to these components using numbers 0, 1 and 2 respectively.

The local transition matrices P^*_i, $i = 0, 1, 2$:

- Component $macro_0$: as the aggregation technique had no effect on this component, its corresponding local transition matrix does not change, $P^*_0 = P_0$.
- Component $micro_{000}$: the number of derivatives of this component is 45, thus its independent behaviour is captured by a 45×45 transition matrix P^*_1. We omit to represent this matrix because of its size.
- Component $micro_{C0}$: as for the component modelling the macro cell, applying the aggregation technique had no impact on component $micro_{C0}$ and therefore its local transition matrix remains unchanged $P^*_2 = P_5$.

The cooperation matrices $P^*_{i,\alpha}$: The set of cooperation action types $\mathcal{Z}$ is defined as follows:

$$\mathcal{Z} = \{handoff_{in.c},\ handoff_{out.c},\ handoff_{up},\ handoff_{up},\ handoff_{in-up.c},\ handoff_{out-up.c}\}$$

The cooperation matrices for the macrocell and the central microcell components remain the same. However, as our peripheral microcell components have been aggregated into one component $micro_{000}$, we have only one cooperation matrix P^*_{1,α_k} for each action type α_k, $k = 1, \ldots, 6$ in $\mathcal{Z}$. As for the local matrix, these matrices are of size 45×45.

Using these matrices, we can then build the generator matrix as follows:

$$\begin{aligned}
Q^* = \bigoplus_{i=0}^{2} P^*_i \quad &+ \quad g^*\delta \left(\bigotimes_{i=0}^{2} P^*_{i,\alpha_1} - \bigotimes_{i=0}^{5} \overline{P}^*_{i,\alpha_1} \right) + f\delta \left(\bigotimes_{i=0}^{2} P^*_{i,\alpha_2} - \bigotimes_{i=0}^{2} \overline{P}^*_{i,\alpha_2} \right) \\
&+ \quad \lambda_2 \left(\bigotimes_{i=0}^{2} P^*_{i,\alpha_4} - \bigotimes_{i=0}^{2} \overline{P}^*_{i,\alpha_4} \right) + \lambda_3 \left(\bigotimes_{i=0}^{2} P^*_{i,\alpha_3} - \bigotimes_{i=0}^{2} \overline{P}^*_{i,\alpha_3} \right) \\
&+ \quad g^*\delta \left(\bigotimes_{i=0}^{2} P^*_{i,\alpha_5} - \bigotimes_{i=0}^{2} \overline{P}^*_{i,\alpha_5} \right) + f\delta \left(\bigotimes_{i=0}^{2} P^*_{i,\alpha_6} - \bigotimes_{i=0}^{2} \overline{P}^*_{i,\alpha_6} \right)
\end{aligned}$$

Unlike f, g^* is a function which depends not only on the current state of component $micro_{000}$, but also on its next state. It is defined as follows:

$$g^*(micro_{ijkl}, micro_{i'j'k'l'}) = \begin{cases} i & \text{if } i' = i - 1 \\ j & \text{if } j' = j - 1 \\ k & \text{if } k' = k - 1 \\ l & \text{if } l' = l - 1 \end{cases}$$

where $micro_{ijkl}$ and $micro_{i'j'k'l'}$ are respectively the current and the next state of the component. Note that in this case the rates could be explicitly incorporated into the expression. We use them only as a notational convenience to keep the expression concise.

8.5. Summary of case study

The introduction of a function into the initial model allows for a more concise model expression in which different apparent rates are captured using a function. It also enables the simpler generator expression, 7.2, to be used.

It is important to recognise that one of the impacts of the application of the aggregation technique is to reduce the number of matrices which form the tensor representation of the model. Thus, in general, the dimension of the product state space captured by the tensor representation will be reduced. This follows since our tensor representation is directly related to the number of components in the model and the aggregation typically amalgamates components. In general the resulting matrices will be larger than the individual ones they replace, but they will still be sparse. This suggests that it is beneficial, whenever possible to apply the aggregation technique and possibly reduce the number of components, before generating the tensor representation of the underlying generator matrix.

Prefix

$$\frac{}{(\alpha, e).E \xrightarrow{(\alpha,e)} E}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,e)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,e)} E' \underset{L}{\bowtie} F} \ (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,e)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,e)} E \underset{L}{\bowtie} F'} \ (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,e_1)} E' \quad F \xrightarrow{(\alpha,e_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L) \qquad \text{where } R = \frac{e_1}{f_\alpha(E)} \frac{e_2}{f_\alpha(F)} \min(f_\alpha(E), f_\alpha(F))$$

Choice

$$\frac{E \xrightarrow{(\alpha,e)} E'}{E + F \xrightarrow{(\alpha,e)} E'} \qquad \frac{F \xrightarrow{(\alpha,e)} F'}{E + F \xrightarrow{(\alpha,e)} F'}$$

Hiding

$$\frac{E \xrightarrow{(\alpha,e)} E'}{E/L \xrightarrow{(\alpha,e)} E'/L} \ (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,e)} E'}{E/L \xrightarrow{(\tau,e)} E'/L} \ (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,e)} E'}{A \xrightarrow{(\alpha,e)} E'} \ (A \overset{def}{=} E)$$

Fig. 8. The operational semantics of PEPA

9. Discussion and conclusions

We have compared SAN and PEPA from the perspectives of model construction and Markov process generation. From both perspectives the functional dependencies of SAN offer clear advantages. We subsequently extended PEPA with functional dependencies and demonstrated that these advantages are preserved in the new formalism. Our extension has been kept as conservative as possible, requiring only minimal changes to the definitons of PEPA. For example, we currenly only allow multiplication within rate expressions rather than a more complete set of arithmetic operators. Our reason for this is that the usual mechnisms for manipulating exponential flows within Markov processes (*decomposition* and *superposition*) are expressed via multiplication.

Like all state-based modelling techniques, both formalisms still suffer from the state space explosion problem. Considerable effort in the performance modelling community has been applied to solutions of this problem. Indeed both our formalisms had established ways of meeting its challenges. The Kronecker representation, of SAN, and the aggregation technique, of PEPA, allow us to deal with larger models but by no means all models. Each is particularly powerful with respect to a specific class of models. By providing a spectrum of techniques within a single formalism we are able to offer the modeller a much greater possibility of being able to handle their model since we can choose an optimal technique, or even apply a number of techniques in appropriate combination. Little work has been carried out to study the implications of this latter strategy. Giving PEPA the capability of using both the aggregation technique and the Kronecker representation makes it a suitable framework for studying the interactions between them.

Both these techniques, in common with other model reduction techniques, can be viewed as shifting the problem rather than avoiding it, in the sense that there are still fundamental limits on the size of model which can be analysed. A radically different approach has recently been proposed for PEPA in which a continuous approximation of the discrete state space is made and analysis is carried out via a set of coupled non-linear ordinary differential equations [Hil05a].

Within this work PEPA can be considered to be representative of all the stochastic process algebras with integrated time and action (cf. Sect. 1). What distinguishes each of these formalisms is principally their mechanisms for synchronisation, but the characteristics of PEPA's cooperation are not important here. Thus the same extension could have been taken for any of these formalisms. One of the reasons for choosing PEPA was because it already has well-established, tool-supported model reduction techniques, thus providing a framework in which the interplay between the tensor representation and other techniques could easily be studied.

This work has opened several avenues of possible future work. We would like to undertake a more systematic study to investigate how the strategic use of aggregation may be used to reduce the size of selected components, as well as the number of components. It has already been suggested by Plateau [FPS96] that functional rates may be used to remove components from SAN models. However, this relies on the judgement and expertise of the modeller. In contrast, preliminary results show that for PEPA models it is possible to automatically recognise and replace candidate components [HK06]. Such components are closely related to the previously defined *resource components* [HT99] which were developed in the context of decomposed (product form) solution of the underlying Markov process. The full implications of this relationship are yet to be explored.

Like the tensor representation for SAN models, the tensor representation for PEPA currently does not avoid the representation of *unreachable states*. These are combinations of local states which, given the semantics of the model, could never be exhibited, but which nevertheless are represented because all combinations of local states are considered. Substantial effort has been applied in the area of Kronecker representation of stochastic Petri nets to avoid the consideration of these states [BK02]. In the future we will seek to modify these algorithms to apply to the state space exploration of PEPA models.

Acknowledgments

This work has been supported by project *"Enhanced techniques for PEPA and their application to software performance modelling"* funded by the CNRS and The Royal Society, and the DEGAS(Design Environments for Global ApplicationS) project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing. The authors would like to thank the anonymous referees whose comments helped to improve the paper.

A. Structured operational semantics for PEPA

The semantic rules, in the structured operational style, are presented in Fig. 8; the interested reader is referred to [Hil94] for more details. The rules are read as follows: if the transition(s) above the inference line can be inferred, then we can infer the transition below the line. The notation $f_\alpha(E)$ which is used in the third cooperation rule denotes the apparent rate of α in E.

These rules, depicted with rates specified by expressions are valid for both the case PEPA with functional rates and the case of PEPA with fixed rates. In the latter case, the rate expressions e, e_1 and e_2 should be replaced by constant rates r, r_1 and r_2. Note that when expressions are used the semantics are symbolic as this contributes to the space saving of the Kronecker expression. Expressions for rates in the matrices for individual components may depend on the local state of other components but may be compactly expressed via a function.

B. Representing the PEPA combinators in SAN

In this section, we investigate the translation of a PEPA model to a SAN model and how to build the PEPA model from a given stochastic automata network. For that, we consider the different combinators of PEPA described above and for each one of them find its equivalent representation in the stochastic automata network formalism.

Prefix: Given the two level grammar imposed on PEPA models by ergodicity considerations, prefix combinator only appears within sequential components. Consequently its translation will be to a local event within single automata. As the prefix combinator captures the possibility of performing an activity in the individual components, this would be translated into an event or transition between states.

Choice: As for the prefix combinator, the choice combinator only appears within sequential components and its translation will be to local events within single automata. The choice combinator reflects competition between two possible activities in the PEPA component. In the SAN this will be represented by having two transitions emanating from the same state.

Cooperation: Cooperating activities will be clearly represented by synchronising events. However these are not entirely the same because of the way that rates are assigned to shared transitions in the two formalisms. In SAN each contributed event is treated as representing only the capability of the automaton involved to participate in such an event. It does not carry any information about the *rate* at which the automaton would be willing to carry out the event. Instead the rate of the synchronisation is set once and globally. In contrast in PEPA each contributing activity has a representation within the component containing it. This representation again captures the capability of participating in the event. If the activity is passive this is all that it represents. However it is also possible that the local representation of the activity in the component includes a rate specification, which captures the maximum speed at which this component could undertake this activity. The rate of the activity synchronised across all the participating components is calculated using the semantic rule:

$$\frac{E \xrightarrow{(\alpha,f_1)} E' \quad F \xrightarrow{(\alpha,f_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L), \qquad R = \frac{f_1}{f_\alpha(E)} \frac{f_2}{f_\alpha(F)} \, min(f_\alpha(E), f_\alpha(F))$$

Thus if we have a system of three components, each capable of carrying out a single activity of type α, but each with its own idea of the rate for the activity, this would be represented in PEPA as the components:

$$P \overset{def}{=} (\alpha, r).P \quad Q \overset{def}{=} (\alpha, r/2).Q \quad R \overset{def}{=} (\alpha, r/4).R$$

If we model the system as

$$System \overset{def}{=} (P \underset{\{\alpha\}}{\bowtie} Q) \underset{\{\alpha\}}{\bowtie} R$$

the rate of the synchronised activity α is $r/4$. Dismantling the system we get the apparent activity rates shown in the table below:

$r_\alpha(P)$	r	$r_\alpha(Q)$	$r/2$	$r_\alpha(R)$	$r/4$
$r_\alpha(P \underset{\{\alpha\}}{\bowtie} Q)$	$r/2$	$r_\alpha(P \underset{\{\alpha\}}{\bowtie} R)$	$r/4$	$r_\alpha(Q \underset{\{\alpha\}}{\bowtie} R)$	$r/4$
$r_\alpha(P \underset{\{\alpha\}}{\bowtie} Q) \underset{\{\alpha\}}{\bowtie} R)$	$r/4$				

Representing this system as a SAN we would choose three simple automaton each with a single state and a self-loop, labelled with the synchronising event α. Globally this would be attributed the rate $r/4$ to reflect the behaviour of the PEPA model. However, starting from this model and removing one of the automaton would result in a new system in which two automaton synchronise but with the same rate which has been set globally: $r/4$.

Hiding/Constant: These two PEPA combinators have no equivalent in SAN formalism.

C. Kronecker algebra

Let A be a matrix of size $n \times n$ and B a matrix of size $p \times p$ such that

$$A = (a_{i_1 j_1}) \quad i_1, j_1 \in [1..n] \qquad and \qquad B = (b_{i_2 j_2}) \quad i_2, j_2 \in [1..p]$$

Definition C.1 *The* **tensor product** *of A and B, noted as $A \bigotimes B$, is a matrix C of size $np \times np$ defined as follows:*

$$C = A \otimes B \quad and \quad C = (c_{ij}) \ i, j \in [1..np]$$

where $c_{ij} = a_{i_1 j_1} b_{i_2 j_2}$, $\ i = (i_1, i_2)$ and $j = (j_1, j_2)$.

Definition C.2 *The* **tensor sum** *of A and B, noted $A \bigoplus B$, is a matrix D of size $np \times np$ defined as*

$$D = A \oplus B = A \otimes I_B + I_A \otimes B$$

where I_M is the identity matrix with the dimension of matrix M.

Example Let A and B be two matrices such that $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$ then

$$C = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}$$

and

$$\begin{aligned} D = A \oplus B &= \begin{bmatrix} a_{11}I_B & a_{12}I_B \\ a_{21}I_B & a_{22}I_B \end{bmatrix} + \begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix} \\ &= \begin{bmatrix} a_{11}+b_{11} & b_{12} & a_{12} & 0 \\ b_{21} & a_{11}+b_{22} & 0 & a_{12} \\ a_{21} & 0 & a_{22}+b_{11} & b_{12} \\ 0 & a_{21} & b_{21} & a_{22}+b_{22} \end{bmatrix} \end{aligned}$$

Remark Note that the sum and the product are associative and distributive operators over addition, but they are not commutative.

References

[ACB84] Ajmone Marsan M, Conte G, Balbo G (1984) A class of generalised stochastic petri nets for the performance evaluation of multiprocessor systems. ACM Trans Comput Syst 2(2):93–122

[BBF$^+$03] Benoit A, Brenner L, Fernandes P, Plateau B, Stewart W (2003) The PEPS software tool. In: Proceedings of the 13th international conference on modelling techniques and tools for computer performance evaluation, Illinois, September 2–7 2003, pp 215–234

[BBFP03] Benoit A, Brenner L, Fernandes P, Plateau B (2003) Aggregation of stochastic automata networks with replicas. In: Proceedings of the international conference on the numerical solution of markov chains (NSMC'03), Illinois, September 2–7 2003, pp 215–234

[BG98] Bernardo M, Gorrieri R (1998) A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time. Theor Comput Sci 202:1–54

[BK02] Buchholz P, Kemper P (2002) Efficient computation and representation of large reachability sets of composed automata. Dis Event Dynam Syst: Theor Appl 12:265–286

[Bor06] Bortolussi L (2006) Stochastic concurrent constraint programming. In: Proceedings of workshop on quantitative analysis of programming languages (QAPL) Vienna, April 2006

[Buc94] Buchholz P (1994) Compositional analysis of a markovian process algebra. In: Herzog U, Rettelbach M (eds) Proceedings of the 2nd process algebra and performance modelling workshop

[Buc99] Buchholz P (1999) Hierarchical structuring of superposed GSPNs. IEEE Trans Softw Eng 25(2):166–181

[CGHT99] Clark G, Gilmore S, Hillston J, Thomas N (1999) Experiences with the PEPA performance modelling tools. IEE Softw 146(1): 11–19

[CM96] Ciardo C, Tilgner M (1996) On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Technical Report 96-35, Institute for Computer Applications in Science and Engineering, Hampton, VA, May 1996

[CM99] Ciardo G, Miner AS (1999) A data structure for the efficient kronecker solution of gspns. In: In P. Buchholz editor, Proc. of the 8th International Workshop on Petri Nets and Performance Models (PNPM'99), Saragoza, Spain, pp 22–31

[Day98] Dayar T (1998) Iterative methods based on splittings for stochastic automata networks. Eur J Oper Res 110:166–186

[DHHR95] Donatelli S, Hermanns H, Hillston J, Ribaudo M (1995) GSPN and SPA compared in practice: modelling a distributed mail system. In: Baccelli F, Jean-Marie A, Mitrani I (eds) Quantitative methods in parallel systems, p 38–51. Springer, Berlin Heidelberg New York

[DHKK01] D'Argenio P, Hermanns H, Katoen J-P, Klaren R (2001) MoDeST – a modelling and description language for stochastic timed systems. In: Process algebra and probabilistic methods, performance modeling and verification: Joint international workshop, PAPM-PROBMIV, Aachen, Germany, September 2001. LNCS, vol 2165, pp 87–104. Springer, Berlin Heidelberg New York

[DHR95] Donatelli S, Hillston J, Ribaudo M (1995) A comparison of performance evaluation process algebra and generalized stochastic Petri nets. In: Proceedings of the 6th Petri Nets and Performance Models Workshop, October 1995, pp 158–168. IEEE Computer Society Press

[DHW04] DiPierro A, Hankin C, Wiklicky H (2004) Continuous-time probabilistic KLAIM. In: Proceedings of SECCO 2004, Electronic Notes in Theoretical Computer Science

[DK01] Donatelli S, Kemper P (2001) Integrating synchronization with priority into a kronecker representation. Perform Evaluat 44(1–4):73–96

[DLM05] DeNicola R, Latella D, Massink M (2005) Formal modelling and quantitative analysis of KLAIM-based mobile systems. In: Proceedings of SAC'05

[Don94] Donatelli S (1994) Superposed generalised stochastic Petri nets: definition and efficient solution. In: Silva M (ed) Proceedings of the 15th international conference on application and theory of Petri nets

[FKV00] Fourneau JM, Kloul L, Valois F (2000) Performance evaluation of a hierarchical cellular network using PEPA. Technical Report RR 2000/2, Laboratoire PRiSM, University of Versailles

[FKV02] Fourneau JM, Kloul L, Valois F (2002) Performance evaluation of a hierarchical cellular network using PEPA. Perform Evaluat 50:83–99

[FPA98] Fernando P, Plateau B, Atif K (1998) Efficient descriptor–vector multiplications in stochastic automata networks. JACM 3:381–414

[FPS96] Fernando P, Plateau B, Stewart WJ (1996) Numerical iusses for stochastic automata networks. In: Ribaudo M, (ed) Proceedings of the fourth process algebra and performance modelling workshop, pp 215–234. CLUT

[GHKR03] Gilmore S, Hillston J, Kloul L, Ribaudo M (2003) PEPA nets: a structured performance modelling formalism. Perform. Evaluat 54(2):79–104

[GHR93] Götz N, Herzog U, Rettelbach M (1993) Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras. In: Performance'93

[GHR97] Gilmore S, Hillston J, Recalde L (1997) Elementary structural analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh

[GHR01] Gilmore S, Hillston J, Ribaudo M (2001) An efficient algorithm for aggregating PEPA models. IEEE Trans Softw Eng 27(5): 449–464

[GP95] Groote JF, Ponse A (1995) The syntax and semantics of μ CRL. In: Ponse A, Verhoef C, van Vlijmen SFM (eds) Algebra of communicating processes '94, workshops in computing series, pp. 26–62. Springer, Berlin Heidelberg New York

[Her99] Hermanns H (1999) Interactive Markov chains. PhD thesis, Erlangen-Nurnberg University

[Hil94] Hillston J (1994) A Compositional Approach to Performance Modelling. Phd. Thesis, University of Edinburgh, 1994

[Hil95] Hillston J (1995) Compositional Markovian modelling using a process algebra. In: Stewart WJ, (ed) Numerical solution of Markov chains. Kluwer

[Hil05a] Hillston J (2005) Fluid flow approximation of pepa models. In: Second international conference on the quantitative evaluation of systems, Torino, Italy, September 2005, pp 33–42. IEEE Computer Society Press

[Hil05b] Hillston J (2005) Tuning systems: from composition to performance. Comput J. The Needham Lecture

[HK01] Hillston J, Kloul L (2001) An efficient kronecker representation for pepa models. In: Proceedings of the joint international workshop, PAPM-PROBMIV 2001, LNCS, vol 2165, pp 120–135, Aachen, Germany. Springer, Berlin Heidelberg New York

[HK06] Hillston J, Kloul L (2006) A function-equivalent components based simplification technique for pepa models. In: Horváth A, Telek M (eds) Formal methods and stochastic models for performance evaluation, Third European performance engineering workshop (EPEW) Budapest, Hungary, June 21–22 2006. LNCS, vol 4054, pp 16–30. Springer, Berlin Heidelberg New York

[HPTvD90] Henderson W, Pearce CEM, Taylor PG, van Dijk NM (1990) Closed queueing networks with batch services. Que Sys 6:59–70

[HRRS01] Hillston J, Recalde L, Ribaudo M, Silva M (2001) A comparison of the expressiveness of SPA and bounded SPN models. In: Haverkort B, German R (eds) Proceedings of the 9th international workshop on Petri nets and performance models, Aachen, Germany, September 2001. IEEE Computer Science Press

[HT90] Henderson W, Taylor PG (1990) Product form in networks of queues with batch arrivals and batch services. Que Syst 6:71–88

[HT99] Hillston J, Thomas N (1999) Product form for a class of PEPA models. Perform Evaluat 35:171–192

[ISO98] ISO/IEC JTCI/SC33. ISO/IEC FCD 15437—Enhancements to LOTOS, May 1998

[JL82] Jacobson S, Lazowska E (1982) Analysing queueing networks with simultaneous resource possession. Commun ACM, 25(2):142–151

[Kem96] Kemper P (1996) Numerical Analysis of Superposed GSPNs. IEEE Trans Softw Eng 22(9):615–628

[KNP02] Kwiatkowska M, Norman G, Parker D (2002) PRISM: Probabilistic symbolic model checker. In: Proceedings of 12th international conference on modelling tools and techniques for computer and communication system performance evaluation, London, UK, April 2002. LNCS, vol 2324, pp 200–204. Springer, Berlin Heidelberg New York

[Mol82] Molloy MK (1982) Performance analysis using stochastic petri nets. IEEE Trans Comput 31(9):913–917

[PF91] Plateau B, Fourneau JM (1991) A methodology for solving markov models of parallel systems. J Parallel Distrib Comput

[PFL88] Plateau B, Fourneau JM, Lee KH (1988) PEPS: a package for solving complex Markov models of parallel systems. In: Proceedings of the 4th international conference on modelling techniques and tools for computer performance evaluation

[Pla84] Plateau B (1984) De l'Evolution du Parallélisme et de la Synchronisation. PhD Thesis, Université de Paris-Sud, Orsay

[Pla85] Plateau B (1985) On the stochastic structure of parallelism and synchronisation models for distributed algorithms. In: Proceedings of the ACM sigmetrics conference on measurement and modelling of computer systems

[Pri95] Priami C (1995) Stochastic π-calculus. Comput J 38(6). Special Issue: Proceedings of the 3rd process algebra and performance modelling workshop

[Rib95] Ribaudo M (1995) On the relationship between stochastic petri nets and stochastic process algebras. PhD Thesis, Dipartimento di Informatica, Università di Torino, May 1995

[SAP95] Stewart WJ, Atif K, Plateau B (1995) The numerical solution of stochastic automata networks. Euro J Oper Res 86:503–525

[SM91] Sanders WH, Meyer JF (1991) Reduced base model construction methods for stochastic activity networks. IEEE J Select Areas Commun 9(1):25–36

Received 25 November 2002
Revised 4 August 2006
Accepted 16 August 2006 by J. Parrow
Published online 9 November 2006