Simulation Modelling Practice and Theory 17 (2009) 1445–1456

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A unified approach to modelling the performance of concurrent systems

Peter G. Harrison [a,*], Catalina M. Lladó [b], Ramon Puigjaner [b]

[a] Department of Computing, Imperial College London, South Kensington Campus, London SW7 1AZ, UK
[b] Universitat de les Illes Balears, Departament de Ciències Matemàtiques i Informàtica, Ctra de Valldemossa, Km. 7.6, 07071 Palma de Mallorca, Spain

ARTICLE INFO

Article history:
Available online 14 June 2009

Keywords:
Product-forms
Performance modelling
Tool interoperability

ABSTRACT

Quantitative design is crucial to ICT and it is therefore important to integrate performance modelling techniques into support environments that facilitate the correct construction of computer systems. We consider Performance Modelling Interchange Formats (PMIFs), which allow models to be specified in a uniform way and ported to a number of tools that solve them. We focus on extending the class of models describable in a PMIF that can be solved analytically – specifically, yielding a product-form solution for their equilibrium state probabilities. We use an extension of an established theorem, called the 'reversed compound agent theorem' (RCAT) as the basis of the analytical modelling tool into which the extended PMIF feeds models. We describe the RCAT methodology in practical terms, how it is integrated into an extended PMIF, and illustrate our methodology with three examples.

1. Introduction

The tools of performance engineering are *quantitative models*, which predict metrics that reflect some view of performance. There are several modelling techniques, each playing a vital role in different circumstances. An evolving, dynamic system can be considered as passing through a sequence of states that change upon the occurrence of events. Thus, discrete event simulation is the most natural and most general modelling formalism since it is possible to mimic in an abstract model the dynamic behaviour of almost any system. The main issues in simulation are reliable interpretation of the simulation outputs, which are essentially sample paths of an underlying stochastic process, and run-time, which may be prohibitively long in detailed simulations and increase rapidly as tighter confidence bands are imposed.

Hence, where possible, analytical models are preferred, based on a mathematical analysis of the stochastic process that precisely describes the dynamics of the system being modelled. The trouble now is that further approximating assumptions must usually be made to achieve mathematical and numerical tractability. As a rule, the more tractable the model needs to be, the more approximations are necessary. Hence, when using approximate analytical models, validation must be done, often initially mutual validation between analytical and simulation models. It is typical for an analytical model to be validated against simulation in simple scenarios, resulting in increased confidence in both models. Then either model can be used to make predictions in more complex cases. As complexity increases, one of the models will ultimately cease to be numerically tractable, but after validation, confidence is high in the surviving model. It often turns out to be most effective to validate a simulation in simple cases against an (assumed) *exact* analytical model and then use that simulation to validate an approximate analytical model, which is the one intended for practical use.

A stochastic process becomes greatly simplified when it possesses the *Markov property*, which essentially states that, at certain (maybe all) time points, the future evolution of the process depends only on its current state, not on its past history.

* Corresponding author.
E-mail addresses: pgh@doc.ic.ac.uk (P.G. Harrison), cllado@uib.es (C.M. Lladó), putxi@uib.es (R. Puigjaner).

1569-190X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2009.06.003

This is often an intuitively reasonable assumption that might be made when there is no information available about the past, for example. Moreover, in such a Markov process (or *Markov chain*) it may be possible to compute the state probabilities of a modelled system at specified times by a numerically tractable algorithm; queueing network models fall into this category [1,4,14,2]. Product-form solutions for the equilibrium state probabilities in Markov chains that represent a set of interacting stochastic processes lead to efficient algorithms for computing many performance measures of interest [16,1,17,14,7]. Although the class of networks that possess such product-forms – often called 'product-form networks' – is highly restricted, large systems usually cannot be solved unless approximating assumptions are made that yield separable solutions, such as product-forms. What is needed, and what this paper addresses, is a unifying methodology for constructing established product-forms that also has the potential to derive new classes of separable solutions, or at least new particular instances within the classes already known. This is accomplished using a theorem, called 'RCAT", that synthesises reversed processes from which separable equilibrium solutions follow.

Section 2 introduces the concepts underlying the methodology we present, the supporting technical results and notation being summarised in the Appendix. In Section 3, we explain how to apply RCAT in practice and outline a mechanised version of a more general result called ERCAT (for *extended* RCAT) [12]. We illustrate with the quite complex example of a G-network [8] as well as with a non-queueing interaction between two serial processes with critical sections. We describe how the methodology is integrated into the environment of a *Performance Modelling Interchange Format* (PMIF) in Section 4 [22,20] and give examples in Section 5, which obtain product-forms for a G-network that represents task killing signals in speculative parallel computation and for a synchronisation amongst software components with critical, mutually exclusive sections. The paper concludes in Section 6 with a brief discussion of implementation issues and further proposed extensions.

Rather than focusing on the theoretical aspects of RCAT, which are well covered elsewhere, the new contributions of the present paper are its practical application and how it can be used to extend a PMIF environment to support non-queueing models. A PMIF is a common representation for performance model data that can be used to move models among modelling tools. A user of several tools that support the format can create a model in one tool and later move the model to other tools for further work without the need to laboriously translate from one tool's model specification to the other. Concretely, PMIF 2.0 [20] is based on the queueing network model paradigm and an extended version, GPMIF (Generalised PMIF) [9], also allows the specification of non-standard queueing networks, such as G-networks [7], and certain fixpoint solutions. Here, we generalise even further the GPMIF to accommodate the extended RCAT, whilst maintaining compatibility with its previous versions. These enhancements are more fundamental than hitherto (leading to the GPMIF of [9]) in that they incorporate non-queueing nodes, specified at the level of state-transitions – essentially by the generators of their CTMCs. This extended GPMIF/ERCAT combination is what was used to derive the product-forms of Section 5.

2. Cooperations and RCAT

Stochastic process algebra (SPA) is an extension of classical process algebra with time delays and probabilities, aimed at providing performance descriptions of concurrent systems. Those having the Markov property are called *Markovian Process Algebras* (MPAs), e.g. TIPP and PEPA [18,15]. Although the methodology used in this paper requires no particular syntax, nor indeed a process algebraic basis, it is convenient to use a fragment of PEPA, which is the simplest MPA, using just two of its connectives or *combinators*:

- *prefix*, written as a 'dot' between an action and a process, e.g. $(a,\lambda)\cdot P$ which denotes a transition with *type* a and rate λ in a Markov transition graph; and
- *cooperation* between two *components* (themselves processes), written $P \underset{L}{\bowtie} Q$. Here L is a set of actions that may occur in P and Q. Any action having a type in L can occur in P when and only when it occurs in Q simultaneously.

A *choice* between two processes, representing alternate transitions from one state to a number of others, is denoted by multiple definitions, e.g. $P = Q;\ P = R$ rather than $P = Q + R$ as in full PEPA.

Separable solutions and product-forms may be derived directly from the generators (instantaneous transition matrix) of the *reversed process* of a continuous time Markov chain (CTMC), together with its own (forwards) generators [17,10]. The determination of a hierarchical class of reversed processes is based on the Reversed Compound Agent Theorem (RCAT) of [10]. This is a compositional result that defines the reversed CTMC of two synchronised CTMCs at equilibrium in terms of their own respective reversed CTMCs. It thereby provides an alternative methodology, with syntactically checkable conditions, that unifies many product-forms, far beyond those for queueing networks.

The theorem is stated in its most general form in the Appendix, under the name MARCAT, applying to multi-agent cooperations in which networks of any number of processes may cooperate, provided that all synchronisations are between two of the component-processes at a time [12]. Less formally, the theorem states the following for a two-node cooperation:

Theorem 1. *Suppose the following conditions hold, the notation being detailed in the Appendix:*

(1) The reversed rate x_a of every active action a is the same at every instance, given by, as in the original RCAT [10], the solution of the rate equations.
(2) The forward and reversed passive and active transition rates satisfy

$$\sum_{a\in\mathcal{P}^{(i,j)\rightarrow}} x_a - \sum_{a\in\mathcal{A}^{(i,j)\leftarrow}} x_a = \sum_{a\in\mathcal{P}^{(i,j)\leftarrow}\setminus\mathcal{A}^{(i,j)\leftarrow}} \overline{\beta_a^{(i,j)}} - \sum_{a\in\mathcal{A}^{(i,j)\rightarrow}\setminus\mathcal{P}^{(i,j)\rightarrow}} \alpha_a^{(i,j)}$$

Then the reversed process of the cooperation $P \underset{L}{\bowtie} Q$ *is*

$$\overline{P \underset{L}{\bowtie} Q} = \overline{R}\{(\overline{a},\overline{p_a}) \leftarrow (\overline{a},\top) \mid a \in \mathcal{A}_P(L)\} \underset{\overline{L}}{\bowtie} \overline{S}\{(\overline{a},\overline{q_a}) \leftarrow (\overline{a},\top) \mid a \in \mathcal{A}_Q(L)\}$$

where

$$R = P\{\top_a \leftarrow x_a | a \in \mathcal{P}_P(L)\}$$
$$S = Q\{\top_a \leftarrow x_a | a \in \mathcal{P}_Q(L)\}$$

Furthermore, the product-form equilibrium probability for state (i,j) *is* $\pi(i,j) \propto \pi_1(i)\pi_2(j)$ *where* $\pi_k(s)$ *is the equilibrium probability of state* s *in* R,S *for* $k = 1,2$*, respectively.*

Under quite mild conditions, the rate equations (a generalisation of the traffic equations of queueing networks [1,14]) have a solution, which is unique (when it can be normalised) by the uniqueness of the equilibrium probabilities of Markov chains. In particular, it is easy to verify that most queueing networks, certainly all variants of G-networks and BCMP networks, have solutions since they possess a certain strongness property [12].

Verification of the conditions of MARCAT requires that the action types in a cooperation set (L above) be checked component-wise at each of their instances – the number of checks is at worst the product of the numbers of local states in each component. Moreover, the states of a component will usually be parameterised in the process algebraic specification, e.g. a queue of positive length corresponds to the PEPA process P_{n+1} and the empty queue to P_0, giving just two parameterised states. The actual number of checks required is then the product of the numbers of *parameterised* local states; two per component, giving four, in this example.

In important special cases, checking is straightforward or trivial and we now consider two of these.

2.1. Queue-like networks

The original RCAT of [10], extended to multiple cooperations, requires that all passive actions are outgoing from every joint state of the cooperation and all active actions are incoming to every joint state. The conditions of MARCAT are then satisfied vaccuously, which is only to be expected since RCAT itself could have been generalised directly to multiple component cooperations without first introducing the relaxed constraints. In particular, MARCAT applies to all of the standard queueing networks, including G-networks and their extensions [1,7].

The original RCAT is well documented, including the way in which it generates product-forms for these generalised queueing networks in a uniform, mechanisable way – see for example [10–12]. We now focus on another special case of the more general conditions, which we will see leads to some surprising product-forms in networks with blocking.

2.2. Invisible passive actions

An action that leads from a process to itself is called *invisible*, since it has no effect. However, if it can cooperate, it can and does have an effect in a synchronisation. Suppose a passive action a is invisible, leading from some state, $i \in P$ say, to itself. Then, referring to the definition of terms and notation in the Appendix, together with the conditions (A.2), we have the reversed rate of this passive action, $\overline{\beta_a^{(i,j)}} = x_a$ for all states in the synchronising process $j \in Q$. If *all* passive actions are invisible, $\mathcal{P}^{(i,j)\rightarrow} = \mathcal{P}^{(i,j)\leftarrow} = \mathcal{P}^{(i,j)}$, say, and the condition becomes:

$$\sum_{a\in\mathcal{A}^{(i,j)\leftarrow}\setminus\mathcal{P}^{(i,j)}} x_a = \sum_{a\in\mathcal{A}^{(i,j)\rightarrow}\setminus\mathcal{P}^{(i,j)}} \alpha_a^{(i,j)} \tag{1}$$

Note that this equation is always satisfied (vacuously) if $\mathcal{P}^{(i,j)} = L$.

2.2.1. Alternative views of the reversed process

In a cooperation in which *all* passive actions are invisible, the state-transition graph of the cooperation is simply the Cartesian product of those of the participating components with the arcs representing active actions in the cooperation set removed when leaving joint states in which the corresponding passive action is not enabled. In the reversed process of the cooperation, an active action, $\overline{a}$ say, is now invisible and causes the corresponding passive action to proceed with rate x_a, when enabled. But x_a is precisely the rate of the reversed active action a of the forward process. Consequently, at equilibrium, the reversed cooperation (calculated as per RCAT) is equivalent to the parallel cooperation of the reversed, independent, *original* processes with their passive actions removed. By Theorem 1, the equilibrium probability of a valid state (i,j) is then proportional to $\pi_P(i)\pi_Q(j)$ where $\pi_P(\cdot)$ and $\pi_Q(\cdot)$ are the equilibrium probability mass functions for the states of the synchronising processes P and Q, respectively.

2.2.2. Critical sections

Suppose processes $P_1, \ldots, P_m$ have respective subsets of states, $E_1, \ldots, E_m$, which are mutually exclusive in that when the state of P_r is in E_r, P_s cannot enter E_s for $r \neq s$. This is modelled by including in a cooperation set L the set of all (active) action types of P_r, T_r, leading into E_r for $1 \leqslant r \leqslant m$. Correspondingly, an invisible passive action is introduced at every state of P_r not in E_r for every active action type in $T_s, s \neq r$. The invisible actions then implicitly 'guard' access to the exclusive areas in other processes by not being enabled in their own process's exclusive area. In small, special cases, e.g. with processes having just two states, one of which is critical, this yields the result that the equilibrium probability of valid state vector $\vec{n}$ is proportional to $\prod_{i=1}^{n} \pi_{P_i}(n_i)$. Note that renormalisation is necessary because of the omitted, invalid states.

This generalises to a result of Boucherie [3], which states that the above product-form solution holds when a process P in its critical section E_P *blocks* the other processes; i.e. the other processes cannot undergo *any* transitions.

Considering, for simplicity, a two-component cooperation, Eq. (1) is satisfied if

$$\sum_{a \in \mathcal{A}^{(i,j)\leftarrow} \setminus \mathcal{P}^{(i,j)}} x_a = \sum_{a \in \mathcal{A}^{(i,j)\rightarrow} \setminus \mathcal{P}^{(i,j)}} \alpha_a^{(i,j)} \tag{2}$$

for all valid states (i,j). When $i \notin E_1$ and $j \notin E_2$, all passive actions are enabled and the equality is true vacuously. When $i \in E_1$ and $j \notin E_2$, all passive actions in P_2 are enabled and all those in P_1 are not, and the equation states

$$\sum_{a \in \mathcal{A}_{P_2}^{j\leftarrow}} x_a = \sum_{a \in \mathcal{A}_{P_2}^{j\rightarrow}} p_{ia}$$

This is true by the first of Kolmogorov's extended criteria (applied to processes R_i in RCAT) since the reversed rate of each active action type a is x_a. The case $i \notin E_1$ and $j \in E_2$ is similar and $i \in E_1, j \in E_2$ yields an invalid state. This argument extends trivially to an arbitrary number of cooperating processes, whereupon the result of [3] follows.

3. Practical application of the RCAT method

Although an application of RCAT does not require whole reversed processes to be determined in general,[1] it does require the specific reversed rates of the synchronising active actions. These can be computed simply if the equilibrium state probabilities of each component process are known from the standard result that the probability flux between any two states in a stationary CTMC is equal to the flux in the opposite direction between the same two states in the reversed process. Since the forward and reversed processes have the same equilibrium state probabilities, we have the result:

Proposition 2. *The reversed process of a stationary Markov process with generator matrix Q and equilibrium probability vector $\vec{\pi}$ is also a stationary Markov process with generator matrix Q′ defined by*

$$q'_{ij} = \frac{\pi_j q_{ji}}{\pi_i}$$

We now give a simple algorithm for deriving a product-form by applying (E)RCAT and follow this with a non-trivial illustration.

3.1. Generic algorithm

For simplicity, we consider the cooperation $P_1 \underset{L}{\bowtie} P_2$. The treatment is similar for n-way cooperations in MARCAT applications.

(1) From P_k construct R_k by setting the rate of every instance of action $a \in L$ that is passive in P_k to x_a, for $k = 1, 2$ (noting that each a will be passive for only one k);
(2) For each active action type a in R_k, $k = 1, 2$, check that its reversed rate is the same for all of its instances, i.e. for all transitions $i \to j$ it denotes between states i, j in the state-transition graph of R_k. Compute and denote this reversed rate (in $\overline{R_k}$) by

$$\overline{r_a} = \pi_k(i) r_a^i / \pi_k(j)$$

where r_a^i is the specified forward rate of this instance i of action type a;[2]
(3) Noting that the symbolic reversed rate $\overline{r_a}$ will in general be functions of the x_b $(b \in L)$, solve the equations $x_a = \overline{r_a}$ for each $a \in L$ and substitute the solutions for the variables x_a in each R_k;

[1] Note, however, that if a reversed process can be obtained very cheaply, it can be invaluable! For example, the reversed process of an M/M/1 queue is well known to be the same queue, which allows specific rates to be written down immediately.

[2] In fact, if the reversed process of the cooperation is required, the full reversed processes $\overline{R_k}$ must be computed.

(4) Check the enabling conditions (A.2) for each cooperating action in each process P_k. For queueing networks, these are as in the original RCAT, namely that all passive actions be always enabled in both the forwards and reversed processes of each component; for exclusively invisible passive actions they are the conditions (1);
(5) The required product-form for state $\underline{s} = (s_1, s_2)$ is now $\pi(\underline{s}) \propto \pi_1(s_1)\pi_2(s_2)$ where $\pi_k(s_k)$ is the equilibrium probability (possibly unnormalised) of state s_k in R_k.

3.2. Illustrative example

We now consider two synchronising Markov processes with state-transition diagrams shown in Fig. 1. There are no critical sections and so the synchronisation is more general than the one considered in the previous section.

All passive actions are invisible so we write $\mathcal{P}^{(i,j)} = \mathcal{P}^{(i,j)\rightarrow} = \mathcal{P}^{(i,j)\leftarrow}$. Thus $\mathcal{P}^{(1,j)} = \{a\}$, $\mathcal{P}^{(2,j)} = \{b\}$ and $\mathcal{P}^{(3,j)} = \emptyset$ for $j = 1, 2, 3$. The conditions on the active and passive, forward and reversed rates of Theorem 1 require the following at each joint state (i, j), where the 'horizontal' process is in local state i and the 'vertical' process is in local state j:

- In states $(1, j)$, $\mathcal{A}^{(1,j)\leftarrow} \setminus \mathcal{P}^{(1,j)} = \{b\}, \emptyset, \{b\}$ and $\mathcal{A}^{(1,j)\rightarrow} \setminus \mathcal{P}^{(1,j)} = \{b\}, \emptyset, \{b\}$, for $j = 1, 2, 3$, respectively;
- In states $(2, j)$, $\mathcal{A}^{(2,j)\leftarrow} \setminus \mathcal{P}^{(2,j)} = \{a\}, \{a\}, \emptyset$ and $\mathcal{A}^{(2,j)\rightarrow} \setminus \mathcal{P}^{(2,j)} = \{a\}, \{a\}, \emptyset$, for $j = 1, 2, 3$, respectively;

Eq. (1) therefore requires $x_b = \beta_1$ (for state $(1,1)$), $x_b = \beta_2$ (for state $(1,3)$), $x_a = \alpha_1$ (for state $(2,1)$) and $x_a = \alpha_2$ (for state $(2,2)$). Hence we must have $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$. The conditions are satisfied vacuously in all other states. The reversed rate of the active action type a at each of its two instances is:

$$x_a = \frac{\pi_2(1)\alpha_1}{\pi_2(2)} = \frac{\pi_2(2)\alpha_2}{\pi_2(1)}$$

Since $x_a = \alpha_1 = \alpha_2$, it follows that $\pi_2(1) = \pi_2(2)$. Similarly, for type b, we find

$$x_b = \frac{\pi_2(1)\beta_1}{\pi_2(3)} = \frac{\pi_2(3)\beta_2}{\pi_2(1)}$$

Since $x_b = \beta_1 = \beta_2$, we must have $\pi_2(1) = \pi_2(3)$. Summing up, the conditions for a product-form are that $\pi_2(1) = \pi_2(2) = \pi_2(3) = 1/3$, $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$.

Notice that a similar result applies if we extend the horizontal process to any number of states in its birth–death chain, so that, for example, it could represent a finite or infinite queue with any of its states controlling the vertical process, which itself can easily be generalised. Indeed, this approach can be extended simply to deal with product-forms in networks with functional rates [6].

4. Performance model interchange formats

Performance model interchange formats facilitate the interchange of models between different performance modelling tools that support the format. A user of several such tools can create a model in one tool and easily move models to other tools for further investigation.

XML based interchange formats have been specified for queueing network models, LQN (*Layered Queueing Networks*), UML (*Unified Modelling Language*), Petri-nets and other formalisms. These formats are used to specify the model and a set of parameters for one run. In [21] an XML interchange schema extension, called Experiment Schema Extension (Ex-SE), defines a set of model runs and the output desired from them. This extension to an interchange schema provides a means of specifying performance studies that is independent of a given tool paradigm.

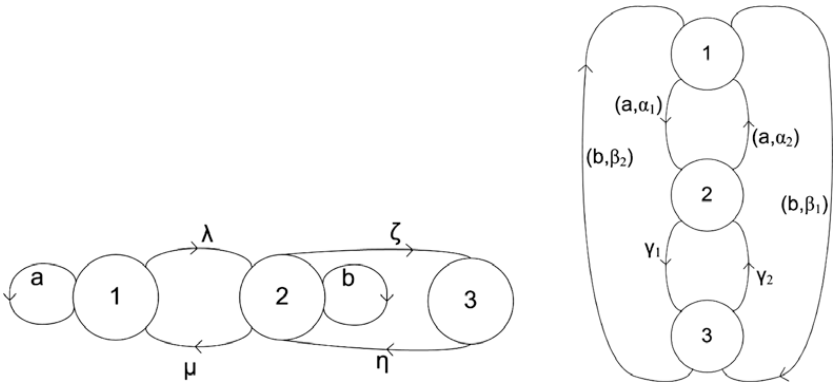

Fig. 1. Synchronised processes with guarded transitions.

In this section we first give an overview of PMIF 2.0 [20] and its generalisation, GPMIF [9] and then detail the GPMIF extension central to this paper.

4.1. PMIF and GPMIF overview

PMIF was originally derived from a meta-model for system performance models that are based on queueing network models (QNMs) [22]. The PMIF meta-model deals with the information that goes into constructing a QNM and was subsequently enhanced, implemented as an XML Schema, and named PMIF 2.0 in [20]. Thus, PMIF 2.0 is a common representation for system performance model data that can be used to move models among modelling tools that use a QNM paradigm. The complete schema is at www.perfeng.com/pmif/pmifschema.xsd.

In summary, the PMIF schema is based on the specification of a *Queueing Network* meta-model in which a *Queueing Network Model* is composed of one or more *Nodes*, zero or more *Arcs*, and one or more *Workloads*. An *Arc* connects one *Node* to another *Node* and several types of *Nodes* may be used. A node of type *Server* provides service for one or more *Workloads*. A *Workload* represents a collection of transactions, tasks or jobs that make similar *Service Requests* from *Servers*. There are two types of Workloads: *Open Workloads* that represent a potentially infinite population of transactions or jobs that arrive from the outside world, receive service at a number of *Servers*, and exit. *Closed Workloads* represent a fixed population of jobs that circulate among the *Servers*. A service request associates the *Workloads* with the *Servers*.

In [9], a generalised node is envisioned as a state-transition system – in a Markov model, by its generator matrix or state-transition graph. However, the GPMIF specification described there did not actually include the generators in this way since the objective was to derive product-forms: if a Markov chain of arbitrary structure were specified, it would rarely yield a product-form.

Instead, the GPMIF specification presented in [9] is an incremental enhancement that accommodates G-network specifications by introducing new workload classes for the negative customers – called *InternalTriggers* and *ExternalTriggers*. It also uses the existing class transition facility to model the change of a positive customer into a negative one, which may occur after a (positive) service completion. The GPMIF meta-model presented in [9] is implemented as an XML Schema and can be found at http://dmi.uib.es/~cllado/GPMIF.html.

4.2. Extensions to GPMIF

The objective of the present extensions is to incorporate restricted classes of non-queueing nodes, specified at the level of state-transitions – essentially by the generators of their CTMC. More concretely, we would like to specify synchronisation amongst software components with critical sections.

Therefore, we introduce a new type of node to the GPMIF meta-model called, *ProcessNode*. We define a Process Node by a pair of lists. The first is a list of (real) numbers (of length n) that represents a finite CTMC with cyclic state-transition graph of n states. The instantaneous transition rate of the ith state in the cycle is equal to the ith element in the given list ($1 \leqslant i \leqslant n$). The second list is a labelled list of lists of state sequence numbers (corresponding to the indices in the first list of rates) that are assigned to the critical section with the given label. For example, $[[5, C1], [7, 8, C2]]$ denotes that a Process Node with at least 8 states has two critical sections labelled $C1$ and $C2$ containing states $[5]$ and $[7, 8]$, respectively. The initial state of a Process Node is defined to be 1. In this model, sequences of states in either a critical or non-critical phase allow restricted non-exponential distributions for the times spent in these phases; e.g. Erlang or a convolution of exponential distributions with different parameters. Coxian distributions could also be defined via an obvious relaxation of the cyclic topology, but we defer this to the general case, where the generator matrix will be supplied.

The implicit semantics of synchronisation between Process Nodes is defined on state-transitions entering into a critical section in one Process Node (an active action) and non-critical states (wrt the same label) in another (invisible passive actions). If a node is in a critical section, every transition in other Process Nodes into a critical state with the same label is blocked; in the case of Boucherie critical sections, the other process will be blocked in *every* state.

4.2.1. Additions to the GPMIF schema

Keeping the schema compatible with PMIF and GPMIF, we added the following new elements to represent Process Nodes, their workloads and services:

- *CriticalSection*, with the attributes *Name* and *Type*, to be able to specify different modes of critical sections, as for example the Boucherie case.
- *ProcessNode*, which has the attributes:
  - *Name*. An identifier for the *ProcessNode*.
  - *NumberOfStates*. The state-graph of a *ProcessNode* is a cycle of length equal to this attribute value. The initial (and final, for workloads that depart to other nodes) state is assumed to be 1, as discussed above.[3]

[3] Generalisation to specification of a *ProcessNode* by a generator matrix is deferred. The numerical parameterisation of this matrix would then be in the *ServiceRequest* type (below), cf. the rates vector in the current version; the state-graph would not necessarily be a cycle.

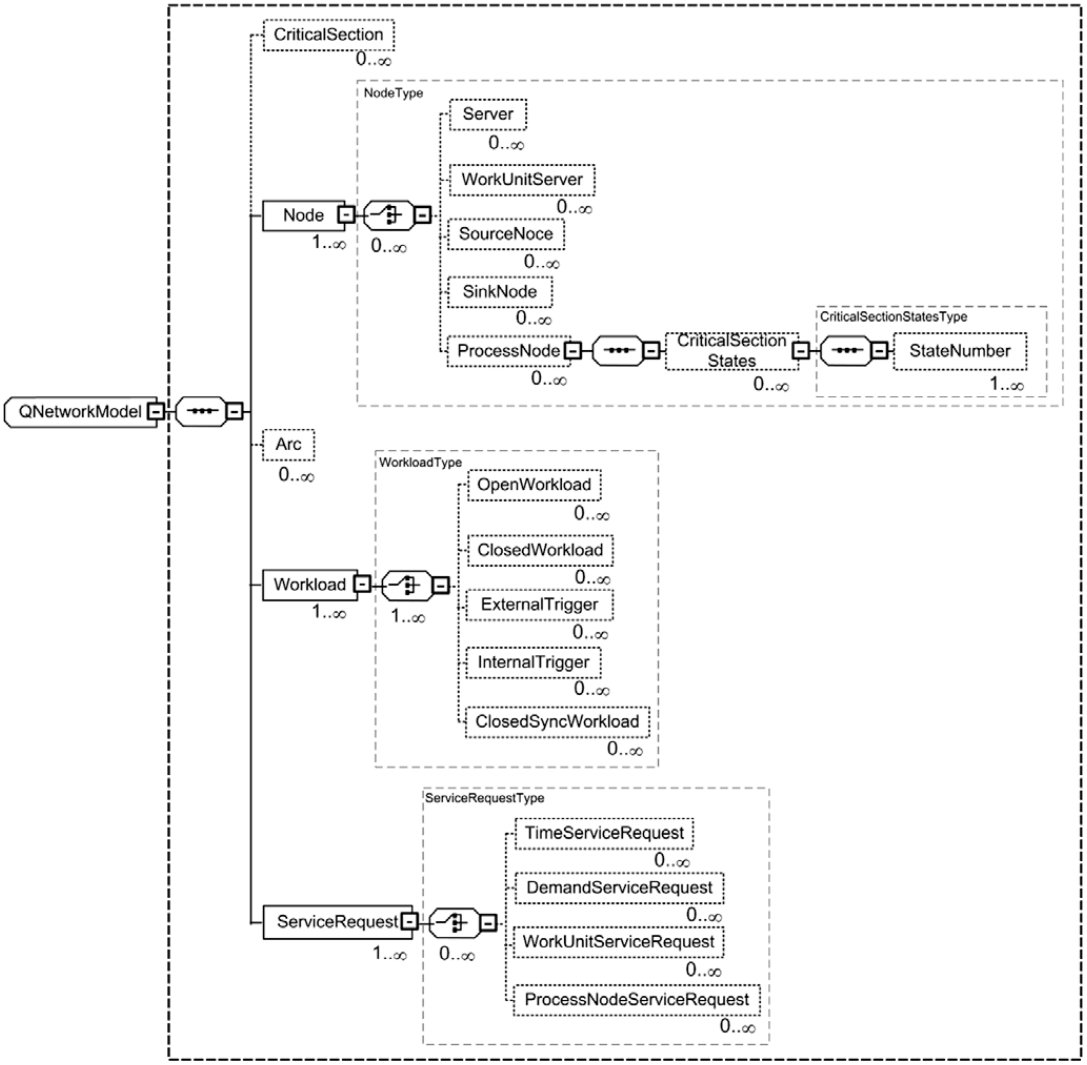

Fig. 2. Extended GPMIF schema.

- *NumberOfCycles*. This is either a constant integer or a probability mass function that specifies the number of cycles a job makes at the node before departing (to a node given by its routing specified in the *ServiceRequestNode* associated with this *ProcessNode* and the job's workload).[4]

  and the sub-element:
- *CriticalSection*. This represents the list of lists of state numbers which are critical, i.e. can only be entered by one synchronising workload (*ClosedSyncWorkload*, defined below) that exclusively uses the critical section associated with the particular (sub)list. Syntactically, it is defined as a list of *CriticalSectionStates* which, in turn, comprises:

  the attribute *CriticalSectionName*, the name of the associated critical section;
  the sub-element *StateNumber*, the list of states of the *ProcessNode* that are exclusive to only one workload associated with the *CriticalSectionName*.

- *ClosedSyncWorkload*. This is the only new workload type, which is required to model Jobs that run in one *ProcessNode* forever. It has the following attributes:
  - *WorkloadName*. An identifier for the workload.
  - *ProcessName*. Jobs of this workload type execute at the *ProcessNode* given by this attribute.
  - *NumberOfJobs* is the number of instances of this *ProcessNode* that synchronise with each other (and maybe other *ProcessNode*s).

Workloads that enter a *ProcessNode* from another node inherit the type *OpenWorkload* or *ClosedWorkload*, depending on whether they enter/leave the network or not. Synchronising workloads that are open in the sense that they arrive at a *ProcessNode* directly from a *SourceNode* similarly inherit the type *OpenWorkload*.

[4] Note that in the more general specification, the number of cycles will be determined by the generator matrix, whereupon the *NumberOfCycles* attribute will be omitted.

- *ProcessNodeServiceRequest*. This type is only associated with a *ProcessNode* (and any workload type) and has the attributes:
  - *ProcessNodeName*, the name of the associated node.
  - *WorkloadName*, the name of the associated workload.
  - *ListOfPhaseRates*, the vector of transition rates used in the *ProcessNode*'s cyclic CTMC. When we generalise to arbitrary CTMCs, this attribute will be the generator matrix.
  - *Transit*, routing probabilities, for departures to other nodes.

The resulting schema is shown in Fig. 2.

5. Case studies

The following two applications illustrate the use of the tool as well as its validation.

5.1. Speculative task killing

Now imagine a computation using speculative parallelism. For example, a distributed database might be searched by several subtasks operating independently on separate segments of the database located on different disks. The computation is complete when the first subtask terminates – here by finding the requested item. At this point 'killing' signals are sent to abort the other subtasks which are no longer required. For simplicity, we consider two such processors running these parallel subtasks, and perhaps other tasks. Each processor is modelled by a queueing node i with service rate μ_i and positive (sub)-task arrival rate λ_i. Completed subtasks send killing signals to the other processor, which are represented by the subtask changing into a negative customer on departing from its current node and passing to the other node. Thus the negative routing probability matrix is

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

and the positive routing probability matrix is $P = 0$. There are no external negative arrivals and so $\Lambda_1 = \Lambda_2 = 0$. Fig. 3 shows this network. Of course, if there are other tasks present that do not participate in the speculative parallelism, they would either leave the system or pass to another node upon completing service, giving non-zero departure or routing probabilities, respectively. We do not consider this case here, however, and hence also have $\lambda_1 = \lambda_2$ since each node will receive the same number of subtasks.[5] In this model we have, for $i = 1, 2, \nu_i = \lambda_i$ and

$$V_1 = \alpha_2 \nu_2; \quad V_2 = \alpha_1 \nu_1$$

where $\alpha_i = \mu_i/(\mu_i + V_i)$. Thus,

$$V_1(\mu_2 + V_2) = \mu_2 \nu_2 = \mu_2 \lambda_2$$

and

$$V_2(\mu_1 + V_1) = \mu_1 \nu_1 = \mu_1 \lambda_1$$

This gives the quadratic equation for V_1:

$$\mu_2 V_1^2 + (\mu_1\mu_2 + \mu_1\lambda_1 - \mu_2\lambda_2)V_1 - \mu_1\mu_2\lambda_2 = 0$$

and similarly for V_2. In the special case that $\lambda_1 = \lambda_2 = \lambda$ and $\mu_1 = \mu_2 = \mu$, writing $V = V_1 = V_2$, we have

$$V^2 + \mu V - \mu\lambda = 0$$

giving the positive solution

$$V = \left(\sqrt{1 + 4\lambda/\mu} - 1\right)\mu/2$$

The product-form solution computed is then:

$$P(n_1, n_2) = (1 - \rho_1)(1 - \rho_2)\rho_1^{n_1}\rho_2^{n_2}$$

where $\rho_1 = \lambda_1/(\mu_1 + V_1), \rho_2 = \lambda_2/(\mu_2 + V_2)$ as required.

Note that, in general, the traffic equations for the negative arrival rates V_i are non-linear and are solved iteratively; here we could obtain a direct solution since the non-linearity was only quadratic.

[5] We do not represent the synchronisation between parallel subtasks either in this simple model. Of course, if other types of arrival were also present, we would expect the impact of this synchronisation to be less and hence our model to be more accurate.

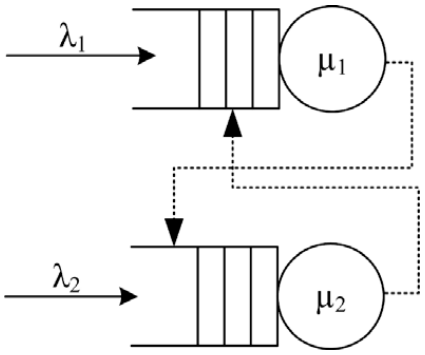

Fig. 3. Speculative killing.

For a numerical example, in the symmetrical case, suppose $\lambda = 21,\ \mu = 16$ so that $V = 3\mu/4 = 12$. Then $\rho = 21/28 = 3/4$ and the solution is

$$P(n_1, n_2) = 0.75^{n_1+n_2}/16$$

This result, a special case, can be used to validate the MARCAT-based tool, which we can then use to calculate solutions in general cases.

5.2. Critical sections

Now consider two processes synchronising over two critical sections C_1, C_2. Both processes are cyclic and represented by the *ProcessNode*s of our extended GPMIF in Fig. 4. Neither process can enter a critical section when the other is in a critical section of the same name, but there are two possible modes of exclusive access when one process is in a critical section (named C, say), as discussed in Section 2.2.2:

(a) either, the other process is completely blocked in that it cannot perform *any* action – Boucherie blocking [3]; or
(b) the other process can perform any action provided it does not enter a critical section of the same name, C.

In the former case, therefore, the naming of critical sections is immaterial since the second process is blocked whatever state it is in. Of course, when there are more than one critical section, a mixture of the modes is possible, varying with the name of the critical section.

For the nodes in Fig. 4, the mean times spent in each state are chosen to be i time units in state i, for $1 \leqslant i \leqslant 6$ in the case of process 1 and $1 \leqslant i \leqslant 4$ for process 2. The MARCAT tool interprets the GPMIF specification and represents the synchronisation between the critical sections using invisible passive actions as described in Section 2.2.2. We first consider Boucherie blocking, for which the conditions (A.2) are satisfied. Notice that in this case, the labelling of the critical sections is unimportant and we could have had a single name $C1 = C2 = C$. Since all the passive actions are invisible, the product-form is

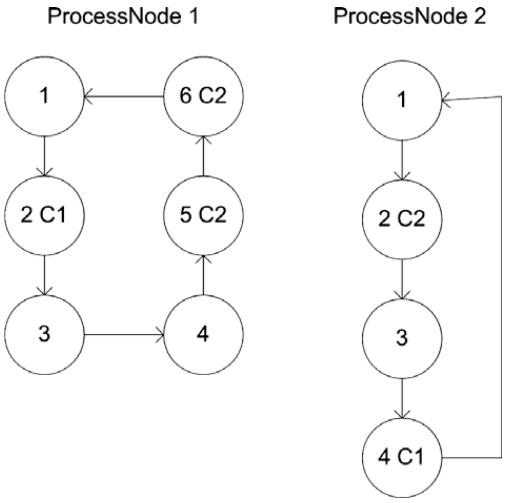

Fig. 4. Two process nodes with critical sections.

easy to find since it is simply the product of the equilibrium state distributions of the individual nodes, renormalised over the set of valid states. The critical states (of either name) in the first process are $\{2,5,6\}$ and in the second $\{2,4\}$. Hence the joint state space, of size 18 states, is $\mathcal{S} = \{(i,j) \mid 1 \leqslant i \leqslant 6, 1 \leqslant j \leqslant 4\} \setminus \{(2,2),(2,4),(5,2),(5,4),(6,2),(6,4)\}$. For a cyclic state-graph, the equilibrium probability of any state is proportional to the reciprocal of its rate, i.e. to its mean holding time. The joint state probabilities are therefore

$$\pi(ij) = \frac{ij}{\sum\limits_{(i,j)\in\mathcal{S}} ij}$$

and we will also use the marginal probabilities $\pi_1(i), \pi_2(j)$.

Using the original raison d'etre of a Performance Modelling Interchange Format, we could also move the model into a stochastic Petri-net tool that solves the underlying Markov chain directly, or the simulation component of the queueing network package QNAP [19]. We calculated the following performance metrics:

- the probability that the first process is in a critical section (so that the second process is blocked), $\pi_1(2) + \pi_1(5) + \pi_1(6) = 0.394$. The 'brute force' Petri-net model gives exactly the same result;
- the probability that the second process is in a critical section (so that the first process is blocked), $\pi_2(2) + \pi_2(4) = 0.364$. Again, the Petri-net model gives exactly the same result. Thus the proportion of time in steady state that the system is blocked, in any way, is 0.758;
- the average time the system spends in a blocked state, before it returns to neither process being blocked, $2\pi_1(2) + 5\pi_1(5) + 6\pi_1(6) + 2\pi_2(2) + 4\pi_2(4) = 3.182$. Of course, the Petri-net model agrees.

Finally, we investigated the second form of exclusive access, (b), where the labelling of the critical sections $C1$ and $C2$ is now significant. Here we found that MARCAT's conditions (A.2) were not satisfied, and indeed a hand calculation revealed that there is no product-form. The Petri-net model gives in this case the results 0.525, 0.473 and 4.203, corresponding to the quantities computed in the above list. Simulation gives 0.589, 0.469 and 4.509, respectively; we suggest the discrepancy is due to too wide a confidence band, but this is not the issue of this paper.

It is interesting to consider the case where the entry state to a critical section (e.g. states 1 and 4 in the first process) has a very short duration. Then, if there are no critical sections, its occupation probability will be very low. Similarly, with Boucherie critical section of type (a) above, because a process cannot progress in any state when the other process is in a critical section, it will rarely be blocked in this 'fast' entry state. However, with blocking of type (b) a blocked process may often enter into this entry state whilst being prohibited from entering the critical section, when held by the other process. Consequently the occupation probability of the entry state may be significantly non-zero. Renormalisation of the state probabilities alone cannot reproduce this effect and we observed large differences when we compared the Boucherie mode result (as a poor approximation) with Petri-net or simulated results for mode (b) when the entry states were fast.

6. Conclusion and future work

The case has been made for performance engineering environments supported by stochastic modelling, akin to methodologies used in traditional engineering disciplines. Such environments should further be integrated into those of software engineering supported by formal methods in theoretical computer science, for example to deal with correctness issues. We have suggested in particular that SPA is a natural unifying formalism for many stochastic modelling methodologies, a claim supported by the use of a PEPA-based MPA to find many classes of product-form solutions through the Multiple Agents Reversed Compound Agent Theorem (MARCAT). This methodology has been mechanised, to a significant extent, to facilitate the uniform derivation of many diverse separable solutions. Current applications range from multi-class queueing networks, through the numerous variants of G-networks, to networks with mutual exclusion and blocking in critical sections [3]. The mechanised solution is currently restricted to actions that can cooperate in only two components at a time, e.g. representing departures from one queue passing to another, but the method of [11] will be applied to handle multiple, instantaneous transitions in chains of components. In addition, it may be possible to model a single active action that can cooperate with several passive actions simultaneously. This would not only induce an alternate approach to triggers but could also account for other types of simultaneous movement of customers in queueing networks, as in [5] for example.

The interchange of performance models between different modelling tools clearly helps the performance engineer since the models can easily be moved to the best suited tool for each application. PMIF and GPMIF are performance model interchange formats used for that purpose. The former is based on the queueing network model paradigm and the latter also allows the specification of G-networks, together with certain more general Markov models. We have here further extended GPMIF to accommodate MARCAT and to enrich it so that it can be used to specify non-queueing behaviours. Specifically, we are now able to specify synchronisation amongst software components with critical sections. Current research has produced a compositional approach to constructing product-forms in Petri-nets and it is planned to include this in the next version of GPMIF [13].

Appendix A. Multi-agent compound agent theorem [12]

In an extension of PEPA, consider now a multiple component, pairwise cooperation $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k (n \geqslant 2)$, where $L = \cup_{k=1}^{n} L_k$ and L_k is the set of synchronising action types that occur in P_k. Every action in each of the n components cooperates with (at most) one other, such that one instance of the action type is active and the other is passive. The semantics of a multiple component cooperation is given in terms of PEPA's dyadic cooperation combinator:

$$\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k = (\ldots((P_1 \underset{M_2}{\bowtie} P_2) \underset{M_3}{\bowtie} P_3) \underset{M_4}{\bowtie} \ldots \underset{M_{n-1}}{\bowtie} P_{n-1}) \underset{M_n}{\bowtie} P_n$$

where $M_k = L_k \cap (\cup_{j=1}^{k-1} L_j)$.

Notation

$\mathcal{P}_k$	the set of *passive* action types in P_k
$\mathcal{A}_k$	the set of *active* action types in P_k
$\mathcal{P}_k^{i\rightarrow}$	the set of action types in L_k that are *passive* in P_k and correspond to transitions *out of* state i in the CTMC of P_k
$\mathcal{P}_k^{i\leftarrow}$	the set of action types in L_k that are *passive* in P_k and correspond to transitions *into* state i in the CTMC of P_k
$\mathcal{A}_k^{i\rightarrow}$	the set of action types in L_k that are *active* in P_k and correspond to transitions *out of* state i in the CTMC of P_k
$\mathcal{A}_k^{i\leftarrow}$	the set of action types in L_k that are *active* in P_k and correspond to transitions *into* state i in the CTMC of P_k
$\mathcal{P}_{\underline{i}}^{\rightarrow}$	the set of action types in $L = \cup_{k=1}^{n} L_k$ that are *passive* and correspond to transitions *out of* state $\underline{i} = (i_1, i_2, \ldots, i_n)$ in the CTMC of $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k$
$\mathcal{P}_{\underline{i}}^{\leftarrow}$	the set of action types in L that are *passive* and correspond to transitions *into* state $\underline{i}$ in the CTMC of $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k$
$\mathcal{A}_{\underline{i}}^{\rightarrow}$	the set of action types in L that are *active* and correspond to transitions *out of* state $\underline{i}$ in the CTMC of $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k$
$\mathcal{A}_{\underline{i}}^{\leftarrow}$	the set of action types in L that are *active* and correspond to transitions *into* state $\underline{i}$ in the CTMC of $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k$
$\alpha_a^{\underline{i}}$	the instantaneous transition rate *out of* state $\underline{i}$ in the CTMC of $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k$ corresponding to *active* action type $a \in L$
$\top_a$	the unspecified rate associated with the action type a in the action $(a, \top_a)$
$\boldsymbol{x}$	the vector $(x_{a_1}, \ldots, x_{a_m})$ of positive real variables x_{a_i} when $L = \{a_1, \ldots, a_m\}$
$\overline{\beta_a^{\underline{i}}(\boldsymbol{x})}$	the instantaneous transition rate *out of* state $\underline{i}$ in the reversed Markov process of $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k\{\top_a \leftarrow x_a | a \in L\}$ corresponding to *passive* action type $a \in L$; note that a is *incoming* to state $\underline{i}$ in the forwards process.

The Multiple Agent Reversed Compound Agent Theorem (MARCAT) defines the reversed process of an n-way cooperation under appropriate conditions, together with a product-form solution for its steady state probabilities, when equilibrium exists. A simplified version, relating to the product-form only, is the following.

Theorem 1 (MARCAT). *Suppose that the cooperation $\underset{\substack{k=1\\L}}{\overset{n}{\bowtie}} P_k$ of components P_k, denoting stationary CTMCs, has a state-transition graph with an irreducible subgraph G and that the cooperation set L is finite. Let $R_k = P_k\{\top_a \leftarrow x_a \mid a \in \mathcal{P}_k\}$ for $k = 1, \ldots, n$. Given that*

(a) every instance of a reversed action, type $\overline{a}$, of an active *action type $a \in \mathcal{A}_k$ has the same rate $\overline{r_a}$ in $\overline{R_k} (1 \leqslant k \leqslant n)$, and $\{x_a\}$ satisfy the rate equations*

$$\{x_a = \overline{r_a} \mid a \in \mathcal{A}_k, 1 \leqslant k \leqslant n\} \tag{A.1}$$

(b) for all joint states $\underline{i} \in G$

$$\sum_{a \in \mathcal{P}_{\underline{i}}^{\rightarrow}} x_a - \sum_{a \in \mathcal{A}_{\underline{i}}^{\leftarrow}} x_a = \sum_{a \in \mathcal{P}_{\underline{i}}^{\leftarrow} \setminus \mathcal{A}_{\underline{i}}^{\leftarrow}} \overline{\beta_a^{\underline{i}}(\mathbf{x})} - \sum_{a \in \mathcal{A}_{\underline{i}}^{\rightarrow} \setminus \mathcal{P}_{\underline{i}}^{\rightarrow}} \alpha_a^{\underline{i}} \tag{A.2}$$

then the cooperation has product-form solution $\pi(\underline{i}) \propto \prod_{k=1}^{n} \pi_k(i_k)$ for the equilibrium probability of state $\underline{i} = (i_1, \ldots, i_n)$, where $\pi_k(i_k)$ is proportional to the equilibrium probability of state i_k in R_k.

The proof verifies Kolmogorov's criteria for reversed processes (as for the original RCAT [10]) and is given in full in [12].

References

[1] F. Baskett, K.M. Chandy, R.R. Muntz, F.G. Palacios, Open, closed and mixed networks of queues with different classes of customers, Journal of ACM 22 (2) (1975) 248–260.
[2] G. Bolch, S. Greiner, H. de Meer, K.S. Trivedi, Queueing Networks and Markov Chains Modeling and Performance Evaluation with Computer Science Applications, Wiley, 2006.
[3] R.J. Boucherie, A characterisation of independence for competing markov chains with applications to stochastic petri nets, IEEE Transactions on Software Engineering 20 (7) (1994) 536–544.
[4] J.P. Buzen, Computational algorithms for closed queueing networks with exponential servers, Communications of the ACM 16 (1973) 527–531.
[5] X. Chao, M. Miyazawa, M. Pinedo, Queueing Networks: Customers, Signals and Product Form Solutions, Wiley, 1999.
[6] J.-M. Fourneau, B. Plateau, W.J. Stewart, Product form for stochastic automata networks, in: Second International Conference on Performance Evaluation Methodologies and Tools, Valuetools 2007, October 2007.
[7] E. Gelenbe, Random neural networks with positive and negative signals and product form solution, Neural Computation 1 (4) (1989) 502–510.
[8] E. Gelenbe, Queueing networks with negative and positive customers, Journal of Applied Probability 28 (1991) 656–663.
[9] Peter G. Harrison, Catalina Lladó, Ramon Puigjaner, A general performance model interchange format, in: First International Conference on Performance Evaluation Methodologies and Tools, Valuetools 2006, Pisa, Italy, ACM International Conference Proceedings, vol. 180, Article No. 6, October 2006, p. 10.
[10] P.G. Harrison, Turning back time in markovian process algebra, Theoretical Computer Science 290 (3) (2003) 1947–1986. January.
[11] P.G. Harrison, Compositional reversed Markov processes with applications to G-networks, Performance Evaluation (December) (2004).
[12] P.G. Harrison, T.T. Lee, Separable equilibrium state probabilities via time reversal in Markovian process algebra, Theoretical Computer Science (2005).
[13] P.G. Harrison, A. Marin, A compositional construction of product-form petri-nets using ERCAT, Technical Report, Department of Computing, Imperial College London and Dipartimento di Informatica, Universit'a Ca+ Foscari di Venezia, 2009.
[14] P.G. Harrison, Naresh M. Patel, Performance Modelling of Communication Networks and Computer Architectures, Addison-Wesley, 1992.
[15] J. Hillston, A compositional approach to performance modelling, PhD Thesis, University of Edinburgh, 1994.
[16] J.R. Jackson, Jobshop-like queueing systems, Management Science 10 (1) (1963) 131–142.
[17] F.P. Kelly, Reversibility and Stochastic Networks, Wiley, 1979.
[18] U. Herzog, N. Götz, M. Rettelbach, Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras, in: Tutorial Proceedings of PERFORMANCE '93, Lecture Notes in Computer Science, vol. 794, Springer-Verlag, 1993.
[19] M. Potier, D. Veran, Qnap2: a portable environment for queueing systems modelling, in: D. Potier (Ed.), International Conference on Modeling Techniques and Tools for Performance Analysis, Paris, May 1984, North Holland, May 1985, pp. 25–63.
[20] C.U. Smith, C.M. Llado, Performance model interchange format (pmif 2.0): xml definition and implementation, in: Proceedings of the First International Conference on the Quantitative Evaluation of Systems, September 2004, pp. 38–47.
[21] C.U. Smith, C.M. Llado, R. Puigjaner, L.G. Williams, Interchange formats for performance models: experimentation and output, in: Proceedings of the Fourth International Conference on the Quantitative Evaluation of Systems, September 2007, pp. 91–100.
[22] C.U. Smith, L.G. Williams, Panel presentation: a performance model interchange format, in: Proceedings of the International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, 1995.