Product-Form Solution in PEPA via the Reversed Process

Peter G. Harrison[1] and Nigel Thomas[2]

[1] Department of Computing, Imperial College London, UK
pgh@doc.ic.ac.uk
[2] School of Computing Science, Newcastle University, UK
nigel.thomas@ncl.ac.uk

Abstract. In this paper we use the reversed process to derive expressions for the steady state probability distribution of a class of product-form PEPA models. In doing so we exploit the *Reversed Compound Agent Theorem* (RCAT) to compute the rates within reversed components of a model. The class of model is, in essence, a generalised, closed, queueing network that might also be solved by mean value analysis, if full distributions are not needed, or approximated using a fluid flow approximation. A general formulation of RCAT is given and the process is illustrated with a running example, including several new variations that consider effects such as multiple servers, competing services and functional rates within actions.

1 Introduction

Quantitative methods are vital for the design of efficient systems in ICT, communication networks and other logistical areas such as business processes and healthcare systems. However, the resulting models need to be both accessible to the designer, rather than only to the performance specialist, and efficient. A sufficiently expressive formalism is needed that can specify models at a high level of description and also facilitate separable and hence efficient mathematical solutions. Stochastic process algebra (SPA) is a formalism that has the potential to meet these requirements.

One approach to tackling the state space explosion problem common to all compositional modelling techniques is through the exploitation of, so called, *product-form solutions*. Essentially, a product-form is a decomposed solution where the steady state distribution of a whole system can be found by multiplying the marginal distributions of its components. The quest for product-form solutions in stochastic networks has been a major research area in performance modelling for over 30 years. Most attention has been given to queueing networks and their variants such as G-networks [12], but there have also been other significant examples, e.g. [1,2,7]. However, these have typically been derived in a rather ad-hoc way: guessing that such a solution exists, and then verifying that the Kolmogorov equations of the defining Markov chain are satisfied and appealing to uniqueness.

D. Kouvatsos (Ed.): Next Generation Internet, LNCS 5233, pp. 343–356, 2011.

The Reversed Compound Agent Theorem (RCAT) [3] for MPA is a compositional result that finds the reversed stationary Markov process of a cooperation between two interacting components, under syntactically checkable conditions [3,4,5]. From this a product-form follows simply. RCAT thereby provides an alternative methodology that unifies many product-forms, far beyond those for queueing networks. At the time, the original study of product-form G-networks was significantly different from previous product-form analyses since the property of *local balance* did not hold and the traffic equations were non-linear. In contrast, the RCAT-based approach goes through unchanged – the only difference is that there are co-operations between one departure transition type and another, as well as between departure transitions and arrival transitions as in conventional queueing networks.

In this paper we use the RCAT approach to find product-form solutions to a class of PEPA models. This class has previously been shown to be amenable to solution by mean value analysis [10] (for expected values only, of course) and fluid flow approximation [11]. A similar model, expressed quite differently, was included in [4].

The context of the present work is as follows:

- Models are expressed explicitly using the full PEPA syntax.
- Reversed components and models are also fully expressed using full PEPA.
- The class of model considered uses active-active co-operation, not currently supported under RCAT or its extensions.
- The class includes the 'counting' approach to specifying (large) groups of identical components in parallel.

2 General RCAT Algorithm

Whilst dependent on reversed processes for its original derivation, an *application* of RCAT can be done purely mechanically if the steady state probabilities are known for the component-processes; from these the reversed processes could be computed if desired, as discussed in the next section. For simplicity, we consider the cooperation $P_1 \underset{L}{\bowtie} P_2$; the treatment is similar for n-way cooperations.

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 $k = 1, 2$ (note 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 a certain quantity $\overline{r_a}$ is the same for all of its instances, i.e. for all transitions $i \to j$ that a denotes in the state transition graph of R_k. This quantity is computed as
$$\overline{r_a} = \pi_k(i) r_a^i / \pi_k(j)$$
where r_a^i is the specified forward rate of the (any, if more then one) instance of action type a going out of state i (must be the same value for all i);
3. Noting that the symbolic reversed rate $\overline{r_a}$ will in general be a function 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;

4. Check the enabling conditions (detailed in [4], but not part of the specific focus of this paper) for each co-operating action in each process P_k. For queueing networks, these are as in the original RCAT, namely that all passive actions be enabled in all states and that all states also have an incoming instance of every active action;
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 (which may be unnormalised) of state s_k in R_k.

For irreducible closed networks, this product-form can always be normalised to give the required steady state probabilities. For irreducible open networks, a separate analysis of ergodicity is required. Notice that all synchronisations in RCAT are between active and passive pairs of actions. In what follows we will relax this condition somewhat.

3 Reversed Processes and Product-Form

RCAT depends on properties of the reversed process. Essentially a reversed process is one that would be observed if time were reversed. For every stationary Markov process, there is a reversed process with the same state space and the same steady state probability distribution, i.e. $\pi_i = \pi'_i$, where π_i and π'_i are the steady state probabilities of being in state i in the forward and reversed process respectively. Furthermore, the forward and reversed processes are related by the transitions between states; there will be a non-zero transition rate between states j and i in the reversed process, $q'_{j,i}$ iff there is a non-zero transition rate between states i and j in the forward process, $q_{i,j}$. A special case is the *reversible* process, where the reversed process is stochastically identical to the forward process, so that $q'_{j,i} = q_{i,j}$; an example is the M/M/1 queue.

The reversed process is easily found if we already know the steady state probability distribution (see Kelly [9] for example). The forward and reversed probability fluxes balance at equilibrium, i.e.

$$\pi'_i q'_{i,j} = \pi_j q_{j,i}$$

and so, since $\pi_i = \pi'_i$, we find:

$$q'_{i,j} = \frac{\pi_j q_{j,i}}{\pi_i}$$

In practical cases we do not already know the steady state distribution for the forward process, and if we did we'd have little need to find the reversed process. However, we can use this result directly to find reversed components within a model (if the components are relatively small), or we could guess the possible reversed rates and use this result as a check.

RCAT uses a simpler methodology to derive the reversed components at the syntactic level, based on finding cycles within a component description. Essentially every choice operator in PEPA introduces a new cycle if the successor

behaviours on each side are different. For a given sequential component S, we can compute the following:

1. Define q_i to be the total outgoing rate from a state i (component behaviour). The conservation of outgoing rate (the first of Kolmogorov's criteria, [9]) gives $q'_i = q_i$, for all behaviours i.
2. Find a covering set of cycles.
3. For each cycle apply the second of Kolmogorov's criteria to give a system of non-linear equations that uniquely define the set of rates in the reversed sequential component $\overline{S}$.

For example, consider the following sequential PEPA component.

$$Task_1 \stackrel{def}{=} (read, \xi).Task_2$$
$$Task_2 \stackrel{def}{=} (compute, (1-p)\mu).Task_1 + (compute, p\mu).Task_3$$
$$Task_3 \stackrel{def}{=} (write, \eta).Task_1$$

There are two cycles.

1. from $Task_1$ to $Task_2$ and back to $Task_1$
2. from $Task_1$ to $Task_2$ to $Task_3$ and back to $Task_1$

Thus, using Kolmogorov's criteria we can compute the following.

$$q'_{1,2}q'_{2,1} = q_{1,2}q_{2,1} = \xi(1-p)\mu$$
$$q'_{1,3}q'_{3,2}q'_{2,1} = q_{1,2}q_{2,3}q_{3,1} = \xi p\mu\eta$$

Furthermore, we know that

$$q'_1 = q'_{1,2} + q'_{1,3} = q_1 = q_{1,2} = \xi$$
$$q'_2 = q'_{2,1} = q_2 = q_{2,1} + q_{2,3} = \mu$$
$$q'_3 = q'_{3,2} = q_3 = q_{3,1} = \eta$$

Hence, the reversed component $\overline{Task_i}$ is easily computed.

$$\overline{Task_1} \stackrel{def}{=} (read, (1-p)\xi).\overline{Task_2} + (read, p\xi).\overline{Task_3}$$
$$\overline{Task_2} \stackrel{def}{=} (compute, \mu).\overline{Task_1}$$
$$\overline{Task_3} \stackrel{def}{=} (write, \eta).\overline{Task_2}$$

For clarity, we illustrate the states of the forward and reversed components in Figure 1.

Once we know the rates in both the forward and reversed processes, we can compute the steady state probabilities using a simple chain rule:

$$\pi_j = \frac{\pi_j}{\pi_{j-1}} \frac{\pi_{j-1}}{\pi_{j-2}} \cdots \frac{\pi_1}{\pi_0} \pi_0$$

Given a sequence of actions from some source state we can therefore compute the steady state probability of being in the resultant target state j in relation to the source state 0 as follows:

$$\pi_j = \frac{q_{0,1}q_{1,2}\cdots q_{j-1,j}}{q'_{j,j-1}\cdots q'_{2,1}q'_{1,0}}\pi_0$$

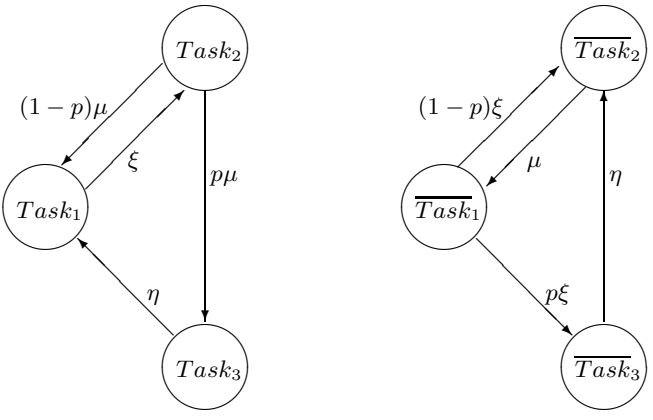

Fig. 1. States of the forward and reversed components

4 A Closed Queueing Model

Now consider a model of a closed queueing network of N jobs circulating around a network of M service stations, denoted $1, \ldots, M$; each station is either a queueing station or an infinite server station, where the number of queueing stations is M_q. Let $\mathcal{M}_q$ be the set of all queueing stations and at each one there is an associated queue (bounded at N) operating a FCFS policy and one server. The servers are able to serve jobs of only one type; each job type, j, is served at rate r_j by queueing station j. At each infinite server station, i, jobs of type i experience a random delay with mean $1/r_i$. All services are negatively exponentially distributed.

There are M job types. For each specified job type, there is exactly one station (either queueing station or infinite server station) which may serve it. When a job of type j completes a service at a given station, it will proceed to service at a station (possibly the same station) as a job of type k according to some routing probability p_{jk}.

In PEPA a queueing station, i, can be modelled as follows.

$$QStation_i \stackrel{def}{=} (service_i, r_i).QStation_i$$

Note that r_i is always specified as finite, and not $\top$. This is because passive actions are subject to the *apparent rate* in PEPA. In addition we impose the restriction that the action enabled at each queueing station is unique to that queueing station, i.e. $\forall i, j \in \mathcal{M}_q$,

$$\mathcal{A}ct(QStation_i) \cap \mathcal{A}ct(QStation_j) = \emptyset$$

The infinite server stations are not represented explicitly.

Each job will receive service from a sequence of stations determined by a set of routing probabilities,

$$Job_i \stackrel{def}{=} \sum_{j=1}^{M} (service_i, p_{ij} r_i).Job_j \ , \ 1 \leq i \leq M$$

where, $0 \leq p_{ij} \leq 1$ and

$$\sum_{j=1}^{M} p_{ij} = 1 \ , \ 1 \leq i \leq M$$

Let $\mathcal{S}_i$ be the set of all job types which perform $service_i$ actions, i.e. $\mathcal{S}_i = j$ if $service_i \in \mathcal{A}(Job_j)$.

The entire system can then be represented as follows:

$$\left(\prod_{\forall i \in \mathcal{M}_{\text{II}}} (QStation_i) \right) \underset{\mathcal{L}}{\bowtie} Job_1[N] \tag{1}$$

where

$$\mathcal{L} = \bigcup_{\forall i \in \mathcal{M}_{\text{II}}} \{service_i\}$$

It is important to note that in this classification the actions which cause the queue station to change mode are not shared actions. The notation $P[N]$ denotes that there are N copies of the component Job_1, but we do not represent each individual in the underlying state space description as would be the case in $Job_1 || \ldots || Job_1$. Instead we are only concerned with the number of components behaving as Job_1, Job_2, etc, at any time. Thus, this representation can be considered to be an explicitly 'lumped' version of $Job_1 || \ldots || Job_1$.

4.1 Reversed PEPA Process

The model presented in the previous section does not meet the existing criteria for RCAT, principally because RCAT is only defined over active-passive cooperation[1]. However, obtaining the reversed process is relatively straightforward since the server components are static. Hence it is only necessary to reverse the single (sequential) job component, which can be done using the most basic result on reversed processes (see Section 3).

This yields a reversed component with the following structure:

$$\overline{Job}_i \stackrel{def}{=} \sum_{k=1}^{M} (service_j, q_{jk} r_j).\overline{Job}_k \ , \ 1 \leq i, j \leq M$$

Where, $0 \leq q_{jk} \leq 1$ and

$$\sum_{k=1}^{M} q_{jk} = 1 \ , \ 1 \leq j \leq M$$

The new routing probabilities, q_{ij}, can be found by applying Kolmogorov's criteria. That is, equate the product of rates around a given cycle in the Job component with the product of the rates around the corresponding reversed cycle

[1] An alternative approach does use RCAT but requires the problem to be mapped into an equivalent problem with functional rates, whereupon the result of [6] can be used

in $\overline{Job}$. Note that $q_{ij} = 0$ in the reversed component iff $p_{ji} = 0$ in the forward component.

The full reversed model can then be specified as

$$\left(\prod_{\forall i \in \mathcal{M}} (QStation_i) \right) \underset{\mathcal{L}}{\bowtie} \overline{Job}_1[N] \tag{2}$$

Where

$$\mathcal{L} = \bigcup_{\forall i \in \mathcal{M}_{\Pi}} \{service_i\}$$

The forward and reversed processes can be used to find a product-form solution (if one exists) by applying Kolmogorov's generalised criteria.

5 Example: A Simple Information Processing System

We now explore the derivation of the reversed process and its use in finding a product-form solution through a specific simple example. Consider the following PEPA specification of a simple information processing system.

$$\begin{aligned} Channel_1 &\stackrel{def}{=} (read, \xi).Channel_1 \\ Process &\stackrel{def}{=} (compute, \mu).Process \\ Channel_2 &\stackrel{def}{=} (write, \eta).Channel_2 \end{aligned}$$

$$\begin{aligned} Task_1 &\stackrel{def}{=} (read, \xi).Task_2 \\ Task_2 &\stackrel{def}{=} (compute, (1-p)\mu).Task_1 + (compute, p\mu).Task_3 \\ Task_3 &\stackrel{def}{=} (write, \eta).Task_1 \end{aligned}$$

The entire system is then specified as

$$(Channel_1 || Process || Channel_2) \underset{\mathcal{L}}{\bowtie} Task_1[N]$$

Items are read from an input channel and processed. If the item meets certain criteria, then it is sent to the output channel before the next item is read, otherwise it is discarded and the next item read.

The reversed component $\overline{Task_i}$ is easily computed as we have already observed.

$$\begin{aligned} \overline{Task}_1 &\stackrel{def}{=} (read, (1-p)\xi).\overline{Task}_2 + (read, p\xi).\overline{Task}_3 \\ \overline{Task}_2 &\stackrel{def}{=} (compute, \mu).\overline{Task}_1 \\ \overline{Task}_3 &\stackrel{def}{=} (write, \eta).\overline{Task}_2 \end{aligned}$$

The system state is described by the triple, $\{i, j, k\}$, where i is the number of components behaving as $Task_1$, j is the number of components behaving as

$Task_2$ and k is the number of components behaving as $Task_3$. Clearly, $i+j+k = N$. We choose $\{N, 0, 0\}$ as a reference state and consider transitions from that state to an arbitrary state $\{i, j, k\}$. Hence we can derive specifications for the steady state probabilities depending on the co-operation set $\mathcal{L}$.

– **All actions are resource limited:** $\mathcal{L} = \{read, compute, write\}$

To reach state $\{i, j, k\}$ there must be $j + k$ *read* actions, each one at rate ξ, followed by k *compute* actions (leading to $Task_3$) at rate $p\mu$. In the reverse process, going from state $\{i, j, k\}$ to state $\{N, 0, 0\}$, there must be k *write* actions at rate η, followed by $j + k$ *compute* actions at rate μ. Hence,

$$\begin{aligned}\pi_{\{i,j,k\}} &= \left(\frac{\xi}{\mu}\right)^{j+k} \left(\frac{p\mu}{\eta}\right)^{k} \pi_{\{N,0,0\}} \\ &= \left(\frac{\xi}{\mu}\right)^{j} \left(\frac{p\xi}{\eta}\right)^{k} \pi_{\{N,0,0\}}\end{aligned}$$

– **No actions are resource limited:** $\mathcal{L} = \emptyset$

If actions are not shared they will occur at a rate proportional to the sum of the number of participants. Thus, when the state is $\{N, 0, 0\}$, the *read* action will occur at rate $N\xi$. Hence, the product of the rates of $j + k$ *read* actions will be $N!\xi^{j+k}/i!$. Similarly, from state $i, j + k, 0$ the product of the rates of k *compute* actions will be $(j + k)!(p\mu)^k/j!$. In the reverse process, starting in state $\{i, j, k\}$, there must be k *write* actions, the product of whose rates is $k!\eta^k$, followed by $j + k$ *compute* actions with rate product $(j + k)!\mu^{j+k}$. Thus,

$$\pi_{\{i,j,k\}} = \frac{N!}{i!j!k!} \left(\frac{\xi}{\mu}\right)^{j} \left(\frac{p\xi}{\eta}\right)^{k} \pi_{\{N,0,0\}}$$

– *compute* **and** *write* **are resource limited:** $\mathcal{L} = \{compute, write\}$

When the cooperation set is a subset of the total action set, we have a situation which lies between the two previous cases. In this case only *read* is a mass action, and in the sequence we consider this is only a factor in the forward transitions. Hence, the $j + k$ read actions occur with rate product $N!\xi^{j+k}/i!$, whereas the k *compute* actions (leading to $Task_3$) each occur at rate $p\mu$. In the reverse process, going from state $\{i, j, k\}$ to state $\{N, 0, 0\}$, there must be k *write* actions at rate η, followed by $j + k$ *compute* actions at rate μ. Hence,

$$\pi_{\{i,j,k\}} = \frac{N!}{i!} \left(\frac{\xi}{\mu}\right)^{j} \left(\frac{p\xi}{\eta}\right)^{k} \pi_{\{N,0,0\}}$$

– **Only** *compute* **is resource limited:** $\mathcal{L} = \{compute\}$

This case is very similar to the previous one, with the exception that the *write* action will be mass action. Hence,

$$\pi_{\{i,j,k\}} = \frac{N!}{i!k!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}$$

5.1 Multiple Service Stations

We now consider the same model components, but with multiple instances of the process component, representing the case where the node has multiple servers.

$$(Input||Process[K]||Output) \underset{\mathcal{L}}{\bowtie} Task_1[N]$$

All components are as defined above. Clearly this extension does not affect the structure of the reversed components, or the solution when $compute \notin \mathcal{L}$. Thus, the reversed model is given by the description,

$$(Input||Process[K]||Output) \underset{\mathcal{L}}{\bowtie} \overline{Task}_1[N]$$

– **All actions are resource limited:** $\mathcal{L} = \{read, compute, write\}$

As in the previous subsection, in going from $\{N, 0, 0\}$ to $\{i, j, k\}$, we will see $j + k$ *read* actions followed by k *compute* actions. As $read \in \mathcal{L}$ each *read* action will occur at rate ξ, However, the rate of *compute* actions will depend on the number of servers and the number of participants. Clearly if $j + k \leq K$ then the system will behave as if $compute \notin \mathcal{L}$, i.e.

$$\pi_{\{i,j,k\}} = \frac{1}{j!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}\ ,\ j + k \leq K$$

If $j > K$ then each forward *compute* action will occur at rate $Kp\mu$. Each reverse *compute* action will occur at rate $K\mu$ until there are fewer than K $\overline{Task}_2$ components.

$$\pi_{\{i,j,k\}} = \frac{1}{K^{j-K}K!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}\ ,\ j > K$$

Finally, if $j + k > K$ and $j \leq K$ then initially *compute* actions will occur at rate $Kp\mu$ (forward) or $K\mu$ (reverse), but once the volume of participants falls below K, then the (cumulative) rate will also fall. In the forward case this means the product of the rates of *compute* actions in the forward process is $K!K^{j+k-K}(p\mu)^k/j!$ and in the reverse it is $K!K^{j+k-K}$. Hence,

$$\pi_{\{i,j,k\}} = \frac{1}{j!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}\ ,\ j + k \leq K$$

– **Only** *compute* **is resource limited:** $\mathcal{L} = \{compute\}$

In the case where the other actions are not shared, the equations are similar as it merely a case of incorporating the cumulative action rates for *read* and *write*.

$$\pi_{\{i,j,k\}} = \frac{N!}{i!j!k!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}\ j+k \leq K$$

$$\pi_{\{i,j,k\}} = \frac{N!}{i!k!K^{j-K}K!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}\ ,\ j > K$$

5.2 Multiple Services from a Station

We consider a variation of this model whereby a job may request service from the same server more than once in a cycle. To do this we will use the same action type and rate in more than one derivative of the job. We do this to avoid a race condition at the server, which would distort the service rate. An alternative approach would be to use a functional rate, which we consider in the next subsection.

Now consider the following model where a shared input/output channel is used.

$$Channel \stackrel{def}{=} (io, \xi).Channel$$
$$Process \stackrel{def}{=} (compute, \mu).Process$$

$$Task_1 \stackrel{def}{=} (io, \xi).Task_2$$
$$Task_2 \stackrel{def}{=} (compute, (1-p)\mu).Task_1 + (compute, p\mu).Task_3$$
$$Task_3 \stackrel{def}{=} (io, \xi).Task_1$$

$$(IO||Process) \underset{\mathcal{L}}{\bowtie} Task_1[N]$$

Clearly the renaming of the *read* and *write* actions (to *io*) has no effect on the structure of the reversed process.

$$\overline{Task}_1 \stackrel{def}{=} (io, (1-p)\xi).\overline{Task}_2 + (read, p\xi).\overline{Task}_3$$
$$\overline{Task}_2 \stackrel{def}{=} (compute, \mu).\overline{Task}_1$$
$$\overline{Task}_3 \stackrel{def}{=} (io, \xi).\overline{Task}_2$$

$$(IO||Process) \underset{\mathcal{L}}{\bowtie} \overline{Task}_1[N]$$

However, the solution of the model will clearly be different, since the rates have been changed and there is potentially more competition for the channel (if *io* is in $\mathcal{L}$).

– **All actions are resource limited:** $\mathcal{L} = \{io, compute\}$

In the forward process, the sequence of actions we consider is not affected by this change, since we are not concerned with io actions from $Task_3$ and there are no components behaving as $Task_3$ when io actions are performed from $Task_1$. However, in the reverse process, we perform k io actions starting in state $\{i, j, k\}$. From the processor sharing-like semantics of PEPA, the rate at which each reverse io action occurs will be dependent on the total number of components behaving as $Task_3$ and $Task_1$. Initially this will be $k\xi/(i+k)$, but will alter as the number of $Task_3$'s decreases. Hence the product of the rate of the k reversed io actions will be $i!k!\xi^k/(i+k)!$. Thus,

$$\pi_{\{i,j,k\}} = \frac{(i+k)!}{i!k!}\left(\frac{\xi}{\mu}\right)^j p^k \pi_{\{N,0,0\}}$$

– **No actions are resource limited:** $\mathcal{L} = \emptyset$

Clearly this case is the same as the initial model with $\mathcal{L} = \emptyset$, with the small modification that the rate η has been replaced by ξ. Thus,

$$\pi_{\{i,j,k\}} = \frac{N!}{i!j!k!}\left(\frac{\xi}{\mu}\right)^j p^k \pi_{\{N,0,0\}}$$

Similarly, the case when only *compute* is resource limited is also the same as previously, since there is no competition on the io action.

– **Only** io **is resource limited:** $\mathcal{L} = \{io\}$

In this case the *compute* action will have a cumulative rate in both the forward and reversed processes, as we have previously observed. Hence,

$$\pi_{\{i,j,k\}} = \frac{(i+k)!}{i!j!k!}\left(\frac{\xi}{\mu}\right)^j p^k \pi_{\{N,0,0\}}$$

5.3 Service Rate Dependent on a Functional Rate

Finally, we consider the case where a station may offer more than one kind of service but the rate at which each service type occurs is governed by a *function* – f_r and f_w below.

$$\begin{aligned} Channel &\stackrel{def}{=} (read, f_r\xi).Channel + (write, f_w\eta).Channel \\ Process &\stackrel{def}{=} (compute, \mu).Process \end{aligned}$$

$$\begin{aligned} Task_1 &\stackrel{def}{=} (read, f_r\xi).Task_2 \\ Task_2 &\stackrel{def}{=} (compute, (1-p)\mu).Task_1 + (compute, p\mu).Task_3 \\ Task_3 &\stackrel{def}{=} (write, f_w\eta).Task_1 \end{aligned}$$

The entire system is then specified as

$$(Channel||Process) \underset{\mathcal{L}}{\bowtie} Task_1[N]$$

where, $\mathcal{L} = \{read, compute, write\}$.

In general, a (so called) *functional rate* can be dependent on properties of the evolution of a model. This does not alter the derivation of the reversed model. In particular, note that the *Channel* component is still considered to be static, despite the choice between actions, and hence the reversed *Channel* component is identical to the forward Channel component.

The reversed model is thus given by the system equation,

$$(Channel||Process) \underset{\mathcal{L}}{\bowtie} \overline{Task_1}[N]$$

where, $\mathcal{L} = \{read, compute, write\}$ and $\overline{Task_1}$ is the reversed component, similar to that derived earlier.

$$\overline{Task_1} \stackrel{def}{=} (read, (1-p)f_r\xi).\overline{Task_2} + (read, pf_r\xi).\overline{Task_3}$$
$$\overline{Task_2} \stackrel{def}{=} (compute, \mu).\overline{Task_1}$$
$$\overline{Task_3} \stackrel{def}{=} (write, f_w\eta).\overline{Task_2}$$

A longer discussion of functional rates in the reversed process was given in [6]. In this instance f_r and f_w can be any functions of i, j and k, but we will restrict ourselves to three simple pairs of functions involving i and k.

- $f_r = \frac{i}{i+k}$, $f_w = \frac{k}{i+k}$

  This is the simple processor sharing function pair and is exactly the same as the previous model, with the exception that there are different rates for *read* and *write*. Thus, if $\mathcal{L} = \{read, compute, write\}$,

$$\pi_{\{i,j,k\}} = \frac{(i+k)!}{i!k!}\left(\frac{\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}$$

- $f_r = ci$, $f_w = ck$, where c is a constant.

  If $c = 1$ this function pair gives the same behaviour as the initial model when $read, write \notin \mathcal{L}$. If $c \neq 1$ there is only a small change.

$$\pi_{\{i,j,k\}} = \frac{N!}{i!k!}\left(\frac{c\xi}{\mu}\right)^j\left(\frac{p\xi}{\eta}\right)^k \pi_{\{N,0,0\}}$$

- $f_r = \frac{i\xi}{i\xi+k\eta}$, $f_w = \frac{k\eta}{i\xi+k\eta}$

  This function pair allocates resource in proportion to service demand. That is, the faster an action is and the more participants it has, relative to the other, the greater the resource that would be allocated. Once more, in the

forward process, the function does not affect the behaviour of the *read* action in the sequence we are interested in (as there are no $Task_3$'s). However, in the reverse process, the rate product is rather more complex:

$$\frac{k!\eta^{2k}}{\prod_{x=1}^{k}(x\eta + i\xi)}$$

Hence,

$$\pi_{\{i,j,k\}} = \frac{\prod_{x=1}^{k}(x\eta + i\xi)}{k!\eta^{2k}} \left(\frac{\xi}{\mu}\right)^{j} \left(\frac{p\xi}{\eta}\right)^{k} \pi_{\{N,0,0\}}$$

6 Conclusions and Further Work

We have used properties of reversed processes to find the steady state probabilities in a class of PEPA models with active-active cooperation. This can be seen as an application of the *Reversed Compound Agent Theorem*, similarly extended to such rate synchronisation. Despite this class lying outside the existing classification of RCAT, the explicit derivation of the reversed process and its use in deriving expressions for the equilibrium distribution is shown to be relatively straight forward. In fact, for reasons of completeness and clarity we have computed the reversed process completely, whereas we have only used a subset of the reversed actions in deriving each product-form solution. Therefore it should be apparent that, in general, it is not necessary to compute all the reversed rates, and in many instances a solution may be derived with only those rates which are most easily found. Indeed, this is the approach adopted for the practical application of the existing RCAT in Section 2.

Of course, we should not be surprised that such results exist for what is, essentially, a closed queueing network. However, through the running example we were able to show the robustness of the solution under a variety of different conditions. Such robustness would clearly be desirable if we were considering a range of different deployment options, for example. There are obviously other extensions to the basic example that we could have considered and some of these could break the product-form solution. For example, we could consider that the server components have alternative modes of operation, which in general would give rise to models without product-form.

Acknowledgements

The authors are supported by the EPSRC funded CAMPA project, which has enabled an extended research visit by Dr Thomas to Imperial College London.

References

1. Balbo, G., Bruell, S., Sereno, M.: Embedded processes in generalized stochastic Petri net. In: Proc. 9th Intl. Workshop on Petri Nets and Performance Models, pp. 71–80 (2001)

2. Boucherie, R.J.: A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. IEEE Trans. on Software Eng. 20(7), 536–544 (1994)
3. Harrison, P.G.: Turning back time in Markovian process algebra. Theoretical Computer Science (January 2003)
4. Harrison, P.G.: Reversed processes, product-forms and a non-product-form. Linear Algebra and Its Applications (July 2004)
5. Harrison, P.G.: Compositional reversed Markov processes, with applications to G-networks. Performance Evaluation (2004)
6. Harrison, P.G.: Product-forms and functional rates. Performance Evaluation 66, 660–663 (2009)
7. Henderson, W., Taylor, P.G.: Embedded Processes in Stochastic Petri Nets. IEEE Trans. on Software Eng. 17(2), 108–116 (1991)
8. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
9. Kelly, F.P.: Reversibility and stochastic networks. Wiley, Chichester (1979)
10. Thomas, N., Zhao, Y.: Mean value analysis for a class of PEPA models. In: Bradley, J.T. (ed.) EPEW 2009. LNCS, vol. 5652, pp. 59–72. Springer, Heidelberg (2009)
11. Thomas, N.: Using ODEs from PEPA models to derive asymptotic solutions for a class of closed queueing networks. In: Proceedings 8th Workshop on Process Algebra and Stochastically Timed Activities. University of Edinburgh, Edinburgh (2009)