PEPA Queues: Capturing Customer Behaviour in Queueing Networks

Ashok Argent-Katwala   Jeremy T. Bradley[1]

Department of Computing
Imperial College London
London, United Kingdom

Abstract

Queueing network formalisms are very good at describing the spatial movement of customers, but typically poor at describing how customers change as they move through the network. We present the PEPA Queues formalism, which uses the popular stochastic process algebra PEPA to represent the individual state and behaviour of customers and servers. We offer a formal semantics for PEPA Queues, plus a direct translation to PEPA, allowing access to the existing tools for analysing PEPA models. Finally, we use the ipc/DNAmaca tool-chain to provide passage-time analysis of a dual Web server example.

Keywords: Stochastic process algebra, PEPA, Queueing networks

1 Introduction

Queueing theory is a well-established discipline, good at describing and analysing quantitatively many complex systems. There are well-known classes of queueing networks which have tractable solutions and many modellers find them a natural modelling formalism to capture resource contention and buffering.

However, while queueing networks are very good at representing the broad structure of a system, they are typically weak at describing the evolution of the individual customers within the network. Indeed, it is quite common to treat all customers as opaque, indistinguishable entities. These customers have no individual internal behaviour but may be segregated into a small set of static classes, when needed.

PEPA Queues [2] augment ordinary queueing networks by allowing customers to have individual behaviour. In this formalism, we use a stochastic process algebra,

---

[1] Email: {ashok,jb}@doc.ic.ac.uk

PEPA, to represent the local behaviour of the customers, the service centres and how they interact. Simple mechanisms from queueing theory describe the migration of customers between queues.

The PEPA Queues formalism is presented with an automatic translation directly into PEPA, giving access to the broad spectrum of existing tools available to analyse systems modelled in PEPA [9,6,12,3].

In Sect. 2 we introduce PEPA and PEPA nets. Sect. 3 gives an overview of what PEPA Queues can do, and how we can model some common mechanisms from queueing theory. Sect. 4 defines the syntax and behaviour of PEPA Queues, with a short example followed by the formal semantics. In Sect. 5, we show how we convert a system of PEPA Queues into an equivalent PEPA model. We show the benefit of maintaining spatial separation and local behaviour using PEPA Queues by modelling a dual Web server on an intranet in Sect. 6. We discuss future work and conclude in Sect. 7.

2   Background

2.1   PEPA

PEPA [11] is a parsimonious stochastic process algebra that can describe compositional stochastic models. As in all process algebras, systems are represented in PEPA as the composition of *components* which undertake *actions*. In PEPA the actions are assumed to have a duration, or delay. Thus the expression $(\alpha, r).P$ denotes a component which can undertake an α action at rate r to evolve into a component P. Here $\alpha \in \mathcal{A}$ where $\mathcal{A}$ is the set of action types and $P \in \mathcal{C}$ where $\mathcal{C}$ is the set of component types. The rate r represents the parameter of an exponential distribution, and the duration is assumed to be a random variable.

PEPA has a small set of combinators, allowing system descriptions to be built up as the concurrent execution and interaction of simple sequential components. The syntax of the type of PEPA model considered in this paper may be formally specified using the following grammar:

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

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a constant which denotes either a sequential component or a model component as introduced by a definition. C_S stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes.

More information including the structured operational semantics for PEPA can be

found in [11]. A brief discussion of the basic PEPA operators is given below:

Prefix The basic mechanism for describing the behaviour of a system with a PEPA model is to give a component a designated first action using the prefix combinator, denoted by a full stop. As explained above, $(\alpha, r).P$ carries out an α action with rate r, and it subsequently behaves as P.

Choice The component $P+Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

Constant It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \stackrel{def}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for example, $X \stackrel{def}{=} (\alpha, r).X$ performs α at rate r forever.

Hiding The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted P/L. Here, the set L identifies those activities which are to be considered internal or private to the component and which will appear as the distinguished unknown type τ.

Cooperation We write $P \underset{L}{\bowtie} Q$ to denote cooperation between P and Q over L. The set which is used as the subscript to the cooperation symbol, the *cooperation set* L, determines those activities on which the components are forced to synchronise. The set L cannot contain the unknown type τ. For action types not in L, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty.

In process cooperation, if a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity* with an appropriate rate. The capacity of a component P to perform an action α is denoted $r_\alpha(P)$, and is termed the *apparent rate*. PEPA respects the definition of *bounded capacity*: that is, a component cannot be made to perform an activity faster by cooperation, so the apparent rate of a shared activity in a cooperation is the minimum of the apparent rates of the activity in the cooperating components.

2.2 PEPA nets

A PEPA net [10] embeds PEPA components within a Petri net, allowing components to cooperate only when they are together on the net. This allows a clear description of systems where different pieces are mobile and move, for example, in and out of an area with network connectivity.

PEPA nets are essentially coloured stochastic Petri nets [1], where the colour of a

token is the state of a PEPA component. This state can change as the system evolves either independently, or in cooperation with their neighbouring components. The places on the net are typed with a component name; they may only be occupied by components that are derivatives of that type. Net-level transitions have associated action-types and rates. Components move in the net when a mobile component performs a net-level transition action, in cooperation with the net itself, to move into an appropriately typed empty slot.

PEPA nets are a good example of a higher level formalism. As well as having their own explicit semantics, they may also be compiled into PEPA, and so analysed with the existing array of PEPA tools. This inspires our approach for PEPA Queues, where we embed PEPA components in a queueing network, rather than a Petri net.

Just as PEPA nets bring the spatial advantages of Petri nets to stochastic modelling, so PEPA Queues offer similar advantages for queueing modellers. Both formalisms allow the expression of greater behavioural control over the tokens or customers.

3   Overview

When building a customer-oriented model of a queueing system in PEPA, it is often tempting to muddle the queueing behaviour with the descriptions of the individual agents. Where we have complex, migrant customers the elegance of the simple cooperations can be obscured by the scaffolding of where each customer is at present.

PEPA Queues encourage the modeller to keep local behaviour and movement separate by representing the system as a network of queues, each of which has a local component, cooperating with the component at the head of the queue.

Note that the state of a PEPA Queue is given by the current state of each of the components waiting in the queue, in order, and the state of the local component. This cannot typically be represented by a vector of buffer occupancies, as would be the case with typical, opaque customers. It is instead akin to having a multi-class queueing network where the class does not influence the queueing discipline.

For example, consider the single PEPA Queue in Fig. 1. It has four places in its input buffer, the local component is Q and it cooperates with the head of the queue over the set of actions L. The *state* of the queue can be given as $[P, P', P, P'']$, where P'' is at the head of the queue. For the moment, we are not concerned with the precise definitions of P, P', P'' and Q, they are ordinary PEPA components. On service, the next customer in the buffer is promoted to replace the served customer.
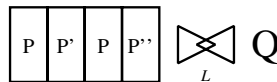

Fig. 1. A single PEPA Queue

For simplicity, we restrict ourselves to closed networks of single-server queues using first-come first-served discipline. The queues all have fixed-size, finite buffers and the

network therefore has a fixed and finite population of customers. There are no losses when destination queues are full; queue services that lead to full queues are blocked. Handling open networks, unbounded buffers, triggers and negative customers is desirable, but the translation into PEPA of these entities is less straightforward. Here we focus on a simple core, which we can build upon in the future.

There are three distinct layers to a PEPA Queues model:

Customers PEPA components represent each individual customer. A customer may evolve internally, or in cooperation with the queue component when it is in service. It may not perform actions in the queue's cooperation set unless it is in service.

Queues Each queue has a local server component which cooperates with the customer in service, over a declared set of actions. By performing routing actions, it determines when customers are despatched and to which queue they are sent. It may also cooperate with the local components at other queues, based on the cooperations defined in the queueing system equation.

Network routing We use PEPA cooperation to decide where components go after being served. Customers may only move to a particular destination queue if there is an open slot in that queue's buffer, and the cooperation of the queue and the customer it is serving offer that routing action.
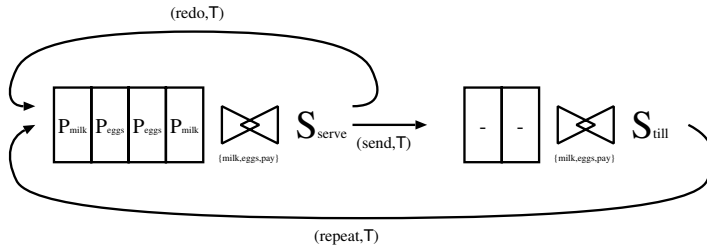

Fig. 2. A small dairy shop, modelled with two PEPA Queues. Queue A comprises the four slot buffer and S_{serve} and queue B the two slot buffer and S_{till}.

The formal syntax for a network of PEPA Queues is given in Sect. 4.1, and the structural operational semantics follow in Sect. 4.3. First, however, we look at a short example.

3.1 **Example:** *Dairy shop*

A system with two queues, A and B, as depicted in Fig. 2. P_{milk} and P_{eggs} are customers who wish to buy milk and eggs respectively. S_{serve} and S_{till} are the server components that hand out goods and take payment. The dashes represent empty slots in B's buffer.

First, we examine the PEPA components that inhabit the network. The P_{milk} and P_{eggs} components are customers who begin waiting in queue A; each is trying to

purchase a different product and pay. The customers can do both actions all the time, and the proper sequence is ensured by how they proceed around the network.

$$P_{eggs} \stackrel{def}{=} (eggs, \top).P_{eggs} + (pay, \top).P_{eggs}$$
$$P_{milk} \stackrel{def}{=} (milk, \top).P_{milk} + (pay, \top).P_{milk}$$

The server components determine the routing of customers in the system. In the first queue the server may be out of stock of the lead customer's item, in which case all the customers must wait for it to become available. This illustrates the interaction of having local state in the server and customer components.

$$S_{eggs} \stackrel{def}{=} (eggs, \top).S_{noeggs} + (expire, e_{eggs}).S_{noeggs}$$
$$S_{noeggs} \stackrel{def}{=} (restockeggs, r_{eggs}).S_{eggs}$$

$$S_{milk} \stackrel{def}{=} (milk, \top).S_{nomilk} + (expire, e_{milk}).S_{nomilk}$$
$$S_{nomilk} \stackrel{def}{=} (restockmilk, r_{milk}).S_{milk}$$

$$S_{wait} \stackrel{def}{=} (eggs, \top).S_{sending} + (milk, \top).S_{sending}$$
$$S_{sending} \stackrel{def}{=} (send, \frac{9r}{10}).S_{wait} + (redo, \frac{r}{10}).S_{wait}$$

$$S_{serve} \stackrel{def}{=} (S_{eggs} \parallel S_{milk}) \underset{\{milk, eggs\}}{\bowtie} S_{wait}$$

$$S_{till} \stackrel{def}{=} (pay, \top).(process, p).(repeat, s).S_{till}$$

It is only when these components perform the queue routing actions (*send*, *redo* and *repeat*) that customers move between queues. These actions may be blocked in three circumstances:

- there are no customers in the queue;
- the routing action is in the queue's cooperation set and the customer in service is not currently offering that action;
- there is no space in the queue the routing action would lead to. Where the same action out of a queue leads to more than one queue, only the ones with space to move into are enabled, and PEPA's competitive choice will select which route we take.

Now, we turn to the queueing network itself. While the picture is useful, we also want a clear, textual form capturing all of the information in the diagram. We need to define both the routing within the network and the state of the individual queues with their buffers. In order to refer to the queues, we give them names. Queue names are prefixed with "Q :", which prevents a clash with any valid PEPA

names, since colons are not permitted in those. The pseudo-component prefixed with "QNet :" is the queueing system, describing the initial state of each queue, their local cooperation set and the system-level cooperation.

Fig. 2 can be fully described by:

$$Q\!:\!A \stackrel{\text{def}}{=} (redo, \top) \rightarrow Q\!:\!A + (send, \top) \rightarrow Q\!:\!B$$
$$Q\!:\!B \stackrel{\text{def}}{=} (repeat, \top) \rightarrow Q\!:\!A$$
$$\text{QNet}\!:\!\text{Sys} \stackrel{\text{def}}{=} (Q\!:\!A[P_{\text{milk}}, P_{\text{eggs}}, P_{\text{eggs}}, P_{\text{milk}}] \underset{\{eggs,milk,pay\}}{\bowtie} S_{\text{serve}})$$
$$|| (Q\!:\!B[-,-] \underset{\{eggs,milk,pay\}}{\bowtie} S_{\text{till}})$$

This syntax is defined in Sect. 4.1.

The first two lines of the specification define the destinations for customers when they are served. Services take place when the server component – possibly in cooperation with the customer – performs any of these routing actions. In this example, all the action names are distinct but a modeller may use competitive choice to choose the destination instead.

We choose to use routing actions which are passive here, and have the server processes, S_{serve} and S_{till}, determine the rates. This is just a modelling choice, and you could have the queueing network determine the rate, or use active-active synchronisation if that better fits the situation being modelled. If a routing action is included in a queue's cooperation set, then the customer in service also influences its routing – the components may only perform the action together, just as with any PEPA cooperation. If they are not in the cooperation, any routing actions the customer performs are purely internal, and do not change the position of the customer within the queueing network.

Note that a properly formed network of PEPA Queues will not have any unresolved passive actions, where a component offers an action passively, but without cooperating over that action. This is the same restriction as in PEPA, but the modeller must also consider that customers move to places where the cooperation set may be different.

The last line of the specification is the queueing system equation. The square brackets contain the state of each queue's buffer, with PEPA component names for customers or a dash representing an empty slot.

If a queue was simply a sink, it would be referred to in other queue routing equations, but not have one of its own, since it has no outbound transitions. There should perhaps be a more elegant way to represent this in the syntax.

For brevity in these descriptions, we also allow $P * n$ and $- * n$ in the queue state description, to represent n copies of P and n spaces respectively. For mechanical simplicity later on, we allow the degenerate constructs of $P * 0$ and $- * 0$, which occupy no space in the queue's buffer.

So, the queueing system equation above could be restated as:

$$\text{QNet} : \text{Sys} \stackrel{\text{def}}{=} (\text{Q} \colon \text{A}[\text{P}_{\text{milk}}, \text{P}_{\text{eggs}} * 2, \text{P}_{\text{milk}}] \underset{\{eggs,milk,pay\}}{\bowtie} \text{S}_{\text{serve}})$$

$$|| \, (\text{Q} \colon \text{B}[- * 2] \underset{\{eggs,milk,pay\}}{\bowtie} \text{S}_{\text{till}})$$

Although we allow cooperations at the network level, they should be used with care. The intention is to use the queueing network to separate local behaviour from spatial migration. Allowing network-level interactions breaks this separation, but can be put to good use for inherently global actions like closing down the whole system, and a variety of forms of reset. Typically we would have, as in the example above, a straight parallel synchronisation between the queues.

3.2   Modelling features

PEPA Queues allow a succinct expression for many queueing mechanisms. For example, we could model breakdown and repair in a network of PEPA Queues as follows, allowing individual servers to fail and be repaired, or with a global reset if all the servers have failed. We omit the definition of the customers, P, for brevity.

$$\text{S}_1 \stackrel{\text{def}}{=} (reprocess, s_1).\text{S}_1 + (continue, s_2).\text{S}_1 + (break, r_{down}).\text{S}_1'$$
$$\text{S}_1' \stackrel{\text{def}}{=} (repair, r_{up}).\text{S}_1 + (allrepair, r_{reset}).\text{S}_1$$

$$\text{S}_2 \stackrel{\text{def}}{=} (route, s).\text{S}_2' + (break, s_{down}).\text{S}_2'$$
$$\text{S}_2' \stackrel{\text{def}}{=} (repair, s_{up}).\text{S}_2 + (allrepair, r_{reset}).\text{S}_2$$

$$\text{S}_3 \stackrel{\text{def}}{=} (route, s).\text{S}_3' + (break, s_{down}).\text{S}_3'$$
$$\text{S}_3' \stackrel{\text{def}}{=} (repair, s_{up}).\text{S}_3 + (allrepair, r_{reset}).\text{S}_3$$

$$\text{Q} \colon \text{Pri} \stackrel{\text{def}}{=} (reprocess, \top) \mapsto \text{Q} \colon \text{Pri} + (continue, \top) \mapsto \text{Q} \colon \text{Sec}$$
$$\text{Q} \colon \text{Sec} \stackrel{\text{def}}{=} (route, \top) \mapsto \text{Q} \colon \text{Ter}$$
$$\text{Q} \colon \text{Ter} \stackrel{\text{def}}{=} (route, \top) \mapsto \text{Q} \colon \text{Pri}$$
$$\text{QNet} \colon \text{Sys} \stackrel{\text{def}}{=} (\text{Q} \colon \text{Pri}[\text{P}, \text{P}] \underset{\emptyset}{\bowtie} \text{S}_1)$$
$$\underset{\{allrepair\}}{\bowtie} (\text{Q} \colon \text{Sec}[-, -, -] \underset{\emptyset}{\bowtie} \text{S}_2)$$
$$\underset{\{allrepair\}}{\bowtie} (\text{Q} \colon \text{Ter}[-, -, -] \underset{\emptyset}{\bowtie} \text{S}_3)$$

4   Defining PEPA Queues

4.1   Syntax

The syntax for PEPA Queues is in two parts: routing between queues and a queueing-system description of the initial state of each queue. All of the components for customers and queue servers are specified in PEPA.

Each queue in a system is given a distinct name, which may be used only once in the final queueing system definition. Queue level routing is defined using an arrow combinator, addressing names of the queues, which all have a "Q:" prefix.

$$\text{QName} ::= \texttt{Q}: \text{Name}$$

$$\text{QRoute} ::= \text{QName} \stackrel{def}{=} \text{QTransList}$$

$$\text{QTransList} ::= \text{QTrans} \mid \text{QTrans} + \text{QTransList}$$

$$\text{QTrans} ::= (\alpha, r) \longrightarrow \text{QName}$$

Our queueing system description uses those Q:X names to define the initial position of the customers in each queue, the cooperation set they have with that queue and the cooperation set amongst the queues:

$$\text{QSys} ::= \texttt{QNet}: \text{Name} \stackrel{def}{=} \text{QList}$$

$$\text{QList} ::= \text{QDef} \mid \text{QDef CoopSet QList}$$

$$\text{QDef} ::= \texttt{Q}: \text{Name [QCustNameList] CoopSet Name}$$

$$\text{CustNameList} ::= \text{CustName} \mid \text{CustName}, \text{CustNameList}$$

$$\text{CustName} ::= \text{Name} \mid - \mid \text{Name} * \text{Num} \mid - * \text{Num}$$

$$\text{CoopSet} ::= \underset{\text{ActNameList}}{\bowtie} \mid \, \|$$

$$\text{ActNameList} ::= \text{Name} \mid \text{Name}, \text{ActNameList}$$

4.2   Behaviour of PEPA Queues

Before defining the formal semantics of PEPA Queues, and describing their mechanical conversion to PEPA, it is illuminating to consider how a network of PEPA Queues may evolve directly.

In any network of PEPA Queues, there are three sorts of transitions that can occur:

Local Any of the server or customer components may evolve independently, provided the action is not in the cooperation set for that queue.

In-service cooperation The customer in service at a queue cooperates with the

stationary component at that queue. If this is a routing action, we treat it as below, otherwise it is treated as a local evolution of both components, at the appropriate combined rate.

Network-level A server transition can cause a customer to move in the network. This happens whenever a stationary component at a queue performs a service action. We move the customer that has been served to the first open slot in the target queue – that is the empty position that is closest to service. This ensures we never leave any blank spaces in a queue's buffer.

Our aim with PEPA Queues is to give the modeller the expressive convenience of queueing models, while retaining much of the simplicity and all of the well-foundedness of PEPA.

We do this by converting PEPA Queues into PEPA. Every PEPA Queue model can be mechanically rewritten as a PEPA model with no change in its behaviour. Some of the structure of the original may be obscured in the compiled PEPA model, so tools will typically prefer to work with the PEPA Queues directly.

Translating to PEPA allows us to use a wide range of tools that already exist, as well as giving toolmakers and modellers who may want to work directly with PEPA Queues a precise definition of exactly how they behave.

Translating to PEPA is not the only way to analyse PEPA Queues. In future, we hope to also offer a translation directly into PEPA nets, preserving much of the spatial structure, which will allow smart PEPA nets tools to exploit spatial features in their analysis. If we create tools that handle PEPA Queues while retaining their structure, then a translation in the other direction – from PEPA nets to PEPA Queues – should be of use too. Further, tools that apply known queueing network results directly, in terms of the PEPA Queues structure, will allow much faster solution of many problems. This is a rich source of future work.

4.3   Semantics of PEPA Queues

Definition 4.1 The relation $\longrightarrow$, operating over the set of component names. $P \xrightarrow{(a,r)} P'$ means that P performs the action a at rate r and evolves into P'. This is as defined in PEPA [11].

Definition 4.2 The relation $\longmapsto$, operating over the set of named queues. $Q{:}A \xrightarrow{(a,r)} Q{:}B$ means there is a queue routing action a at rate r leading from queue A to queue B.

Definition 4.3 The relation $\mapsto\!\!\!\!/\,$, operating over the set of named queues. $Q{:}A \xrightarrow{a}\!\!\!\!/\,$ is equivalent to $\nexists Q{:}B[Q{:}A \xrightarrow{(a,r)} Q{:}B]$, that is a is not a routing action for $Q{:}A$.

Definition 4.4 $r_\alpha(P)$ is the apparent rate function, as defined in [11].

Definition 4.5 $r'_\alpha(A)$ is a secondary apparent rate function, for queue routing actions. α must be a routing action for Q:A and $r'_\alpha(A)$ is the sum of the rates of all the enabled α-activities for Q:A in the present state.

Now we can specify individually all the legal ways a network of PEPA Queues can evolve. Throughout, each ellipsis signifies a (possibly empty) list of components that remains unchanged after the transition. $\mathbb{N}$ denotes the positive integers, so $-*n$ below means that there is at least one empty slot in the queue's buffer.

Customer alone

This rule governs a customer evolving independently, anywhere in the queue. Note that this only allows actions that are not in the queue's cooperation set. Customers that are not in service may not perform those actions, and customers that are in service perform them via rule *Local cooperation*.

$$\frac{P\xrightarrow{(\alpha,\lambda)}P'}{Q{:}A[...,P,...] \bowtie_L S\xrightarrow{(\alpha,\lambda)}Q{:}A[...,P',...] \bowtie_L S}\quad \alpha\notin L$$

Server alone

The server process may perform actions not in the queue cooperation set independently, no matter the state of the buffer.

$$\frac{S\xrightarrow{(\alpha,\lambda)}S'}{Q{:}A[...] \bowtie_L S\xrightarrow{(\alpha,\lambda)}Q{:}A[...] \bowtie_L S'}\quad \alpha\notin L$$

Local cooperation

Only the head of the queue may cooperate with the server process for actions in L, and they must perform them together.

$$\frac{P \bowtie_L S\xrightarrow{(\alpha,\lambda)}P' \bowtie_L S' \quad Q{:}A\not\xrightarrow{\alpha}}{Q{:}A[...,P] \bowtie_L S\xrightarrow{(\alpha,\lambda)}Q{:}A[...,P'] \bowtie_L S'}\quad \alpha\in L$$

Server routing 1

Declares that in order for a customer to move between queues, the server component at the first queue must perform a routing action, leading to a queue which has at least one space. The moved customer moves to the first empty slot. Where n is the largest such n to describe the blank spaces in $Q:B$. Routing actions may not appear in the inter-queue cooperation set, L_Q.

$$\frac{S_A\xrightarrow{(\alpha,\lambda)}S'_A \quad Q{:}A\xrightarrow{(\alpha,\lambda_q)}Q{:}B}{(Q{:}A[...,P] \bowtie_{L_A} S_A) \bowtie_{L_Q} (Q{:}B[-*n,...] \bowtie_{L_B} S_B)\xrightarrow{(\alpha,R)}}{(Q{:}A[-,...] \bowtie_{L_A} S'_A) \bowtie_{L_Q} (Q{:}B[-*(n-1),P,...] \bowtie_{L_B} S_B)}\quad \alpha\notin L_A,\ \alpha\notin L_Q,\ n>0,\ R \text{ as below}$$

Server routing 2

As *Server routing 1* but where the routing action leads to the same queue. Note that we still need an empty slot, to avoid unduly prioritising local routing. Otherwise,

a customer could be allowed to move to the back of the queue it has just left when arrivals from other queues are disallowed.

$$\frac{S_A \xrightarrow{(\alpha,\lambda)} S'_A \quad Q{:}A \xmapsto{(\alpha,\lambda_q)} Q{:}A}{(Q{:}A[-{*}n,...,P] \underset{L_A}{\bowtie} S_A) \xrightarrow{(\alpha,R)} (Q{:}A[-{*}n,P,...] \underset{L_A}{\bowtie} S'_A)} \qquad \alpha \notin L_A,\; n>0,\; R \text{ as below}$$

Coop routing 1

For a routing action that is also in the queue's cooperation set, the action must occur in cooperation between the lead customer and the queue's server. Again, routing actions may not appear in the inter-queue cooperation set, L_Q.

$$\frac{P \underset{L_A}{\bowtie} S_A \xrightarrow{(\alpha,\lambda)} P' \underset{L_A}{\bowtie} S'_A \quad Q{:}A \xmapsto{(\alpha,\lambda_q)} Q{:}B}{\begin{array}{c}(Q{:}A[...,P] \underset{L_A}{\bowtie} S_A) \underset{L_Q}{\bowtie} (Q{:}B[-{*}n,...] \underset{L_B}{\bowtie} S_B) \xrightarrow{(\alpha,R)} \\ (Q{:}A[-,...] \underset{L_A}{\bowtie} S'_A) \underset{L_Q}{\bowtie} (Q{:}B[-{*}(n-1),P',...] \underset{L_B}{\bowtie} S_B)\end{array}} \qquad \alpha \in L_A,\; \alpha \notin L_Q,\; n>0,\; R \text{ as below}$$

Coop routing 2

Just as for *Coop routing 1*, but routing to the same queue. As with *Server routing 2* we require an empty slot in the queue.

$$\frac{P \underset{L_A}{\bowtie} S_A \xrightarrow{(\alpha,\lambda)} P' \underset{L_A}{\bowtie} S'_A \quad Q{:}A \xmapsto{(\alpha,\lambda_q)} Q{:}A}{Q{:}A[-{*}n,...,P] \underset{L_A}{\bowtie} S_A \xrightarrow{(\alpha,R)} Q{:}A[-{*}n,P',...] \underset{L_A}{\bowtie} S'_A} \qquad \alpha \in L_A,\; \alpha \in L_Q,\; n>0,\; R \text{ as below}$$

For the last four rules:

$$R = \frac{\lambda}{r_\alpha(S_A)} \frac{\lambda_q}{r'_\alpha(A)} \min(r_\alpha(S_A), r'_\alpha(A)))$$

where R represents the rate of active cooperation between the service component and the queueing network. As in PEPA, it reflects the rate of the slower component in the cooperation.

Note that we do not allow cooperation between the queues for routing actions. If we did then we would be including a potentially interesting class of synchronised, coupled queues but at the expense of a substantially more complex translation. With our present translation, we use the names of the routing actions, decorated with where they are occurring to provide us with global, unique names for each of the routing actions. If we allowed cooperation at this level, we would need some other mechanism for migrating components. It may be possible to extend still further the information we carry in the expanded action names, but needing to represent which pair of processes is cooperating in the action name would lead to a large explosion in the number of names used. We could conceivably use a synchronous immediate action to achieve these coupled queues, but we would first need to define that concretely.

5   Translation to PEPA

We use a similar approach to that used to translate PEPA nets to PEPA, when translating PEPA Queues to PEPA. We encode the spatial portion of the model by having dormant versions of every component that may occupy a given location, and simply activate them when, in the higher level model, that component arrives in that particular location.

This is somewhat troublesome, since it leads to a large explosion in the number of components in the system, and we typically create both a huge model, textually, and a very large state space. However, it does mean that we can put to work existing tools for handling PEPA models, many of which can handle very large state spaces, with limits that are constantly improving.

Since the slots in our buffers are not typed in any way, we need a definition for each component that may occur, in every position it may occupy.

5.1   Translation detail

Any translation from PEPA Queues to PEPA will in general increase the number of component definitions by a factor of $n(m+1)$, in a system which has n queueing positions and m customer states. We need to distinguish between each of the n queueing slots being empty or having any one of the m customer components in.

With a clever encoding, we may be able to avoid representing unreachable states: for example where a particular customer could never reach part of the network. However, the cost of exploring the reachability, conditioned on the actions it may perform as it traverses the network, will be too high for non-trivial networks.

The basic structure of the translated model is as follows:

(i) Each customer type is represented by many component definitions, one for each spatial state and derivative customer component that the customer could reach. There is one spatial state for each of the queueing positions.

(ii) The queueing transitions are split into a number of distinguishable action names, each leading to a particular slot in a queue. We replace each routing action with a choice of several different, new actions which encode the destination. All the new actions happen at the same rate as the original and we then ensure that only one of them is active at once. When the same action is used to route to different destinations, each is expanded in this manner, and we use PEPA's competitive choice to determine which route is used.

(iii) There is a *population tracker* component for each queue which tracks the number of customers at that queue. The tracker component for a queue X with n slots would have $n+1$ states, $QX_{\mathrm{pop}0}$ to $QX_{\mathrm{pop}n}$. In each state it enables just the queue routing actions to ensure that customers are delivered to the first

available slot, and that no customers may join a full queue. To handle different queueing disciplines we would alter the actions these components enabled.

Note that to be allowed to move, the destination slot must be empty. This means that in the example from Sect. 3.1, when all four customers are in queue A, the *redo* action is disabled. Even though there would be a free slot after the lead customer moves, there is not one until then. If we allowed this form of movement it would give preferential treatment to customers coming from the same queue. That is, a customer would still be able to move to the back of its own queue, while arrivals from other queues were blocked.

The subscripts Xn on the routing actions indicate the current queue length at queue X, including any customer in service. So, on being served by this action, a customer would be placed in slot $n + 1$ of queue X.

The full source of the translation of the dairy shop example can be found in Appendix A. It can be obtained by applying the following transformation.

Let $R_{in}(X)$ be the set of routing actions in to queue X; $R_{out}(X)$ be the routing actions out of queue X and $Capacity(A)$ be the number of places in the buffer at queue A. QueueNames is the set of names of all the queues in the system.

The population trackers for a queue, A, are as follows. The index indicates the number of customers at the queue.

$$\mathrm{QA}_{\mathrm{pop}0} \stackrel{def}{=} \sum_{a \in R_{in}(A)} (a_{A0}, \top).\mathrm{QA}_{\mathrm{pop}1}$$

$$\mathrm{QA}_{\mathrm{pop}i} \stackrel{def}{=} \sum_{a \in R_{in}(A)} (a_{Ai}, \top).\mathrm{QA}_{\mathrm{pop}i+1} \qquad \text{for } 0 < i < Capacity(A)$$

$$+ \sum_{A \stackrel{(d,r)}{\longmapsto} B} \sum_{k=0}^{Capacity(B)-1} (d_{Bk}, \top).\mathrm{QA}_{\mathrm{pop}i-1}$$

$$\mathrm{QA}_{\mathrm{pop}\,Capacity(A)} \stackrel{def}{=} \sum_{A \stackrel{(d,r)}{\longmapsto} B} \sum_{k=0}^{Capacity(B)-1} (d_{Bk}, \top).\mathrm{QA}_{\mathrm{pop}\,Capacity(A)-1}$$

The server components are altered to account for the expanded names of routing actions and to ensure that the local cooperation actions are distinct for each queue. So, for a server component, S, residing at queue A, we make a new component S_A. The local cooperation set at A is L_A. For each action-type, a, that S performs, there are three cases:

(i) $a \notin R_{out}(A) \wedge a \notin L_A$

(ii) $a \notin R_{out}(A) \wedge a \in L_A$

(iii) $a \in R_{out}(A)$

For case 1, we do not need to modify the action at all. For case 2, we add the queue name to the action name, to ensure this action cannot occur when a customer is at a different queue. For case 3, we replace each occurrence of a with a choice of new action-types representing all the destination slots the action could lead to. We denote these four sets of action names L_1, L_2, L_3 respectively.

$$
\begin{aligned}
S_A \stackrel{def}{=} & \sum_{S \xrightarrow{(a,r)} S' \wedge a \in L_1} (a, r).S'_A \\
& + \sum_{S \xrightarrow{(a,r)} S' \wedge a \in L_2} (a_A, r).S'_A \\
& + \sum_{S \xrightarrow{(a,r)} S' \wedge a \in L_3} \sum_{A \xmapsto{a} B} \sum_{k=0}^{Capacity(B)-1} (a_{Bk}, r).S'_A
\end{aligned}
$$

We repeat the above translation for all the derivative states of S.

For each customer component, P, we make a version of that component in each queueing position of each queue. P_{A0} represents the customer, P, in service at queue A; P_{Ai}, for $1 \leq i \leq Capacity(A)$ is customer P waiting in position i of A. When in service, the translated component has all the behaviour of P, but routing actions now lead to the next appropriate queueing position.

We partition the actions a customer performs into three sets at each queue, A:

L_{C1}^A: Actions that are not in the local cooperation set and are not outbound routing actions. $a \notin L_A \wedge a \notin R_{out}(A)$.

L_{C2}^A: Actions that are in the local cooperation set, but are not outbound routing actions. $a \in L_A \wedge a \notin R_{out}(A)$.

L_{C3}^A: Actions that are outbound routing actions. $a \in R_{out}(A)$.

The customer at the front gets to actually perform the actions, and change state. For actions in L_{C1}^A, waiting customers also perform the actions in a straightforward manner. For actions in L_{C2}^A waiting customers cannot perform them, but must passively witness any of the local cooperation actions for any queue. Actions in L_{C3}^A are the most complicated. Waiting customers must passively witness these actions, then move forward to the place in front, but without changing their own local state, irrespective of the original definition of the customer component.

$$
\begin{aligned}
P_{A0} \stackrel{def}{=} & \sum_{P \xrightarrow{(a,r)} P' \wedge a \in L_{C1}^A} (a, r).P'_{A0} \\
& + \sum_{P \xrightarrow{(a,r)} P' \wedge a \in L_{C2}^A} (a_A, r).P'_{A0}
\end{aligned}
$$

$$+ \sum_{\mathrm{P}\xrightarrow{(a,r)}\mathrm{P'}\wedge a\in L^A_{C3}\wedge \mathrm{A}\xmapsto{a}\mathrm{B}} \sum_{k=0}^{Capacity(B)-1} (a_{Bk}, r).\mathrm{P'}_{\mathrm{B}k+1}$$

$$\mathrm{P}_{\mathrm{A}i} \stackrel{def}{=} \sum_{\mathrm{P}\xrightarrow{(a,r)}\mathrm{P'}\wedge a\in L^A_{C1}} (a, r).\mathrm{P'}_{\mathrm{A}i} \qquad\qquad \text{for } 1 \le i \le Capacity(A)$$

$$+ \sum_{\mathrm{X}\in\mathrm{QueueNames}} \sum_{a\in L_X} (a_X, \top).\mathrm{P}_{\mathrm{A}i}$$

$$+ \sum_{\mathrm{P}\xrightarrow{(a,\top)}\mathrm{P'}\wedge a\in L^A_{C3}\wedge \mathrm{A}\xmapsto{a}\mathrm{B}} \sum_{k=0}^{Capacity(B)-1} (a_{Bk}, \top).\mathrm{P}_{\mathrm{A}i-1}$$

We use two different cooperation sets: L_R, with just the routing actions; and L_{Sys} with both the routing actions, and the local cooperating actions for each queue:

$$L_R = \bigcup_{\mathrm{X}\in\mathrm{QueueNames}} \bigcup_{c\in R_{in}(X)} \bigcup_{0\le i < Capacity(X)} \{c_{Xi}\}$$

$$L_{Sys} = L_R \cup \bigcup_{\mathrm{X}\in\mathrm{QueueNames}} \bigcup_{a\in L_X} \{a_X\}$$

We build the final system equation from three major components, representing the state of the queues (QState), the migrant customers (Customers) and the stationary server components (Servers), respecting the initial states of each. Each represents the initial state of the system as described by the queueing network system equation.

Suppose that the queues are named $A, B, \ldots, Z$, that the initial populations at each are $a, b, \ldots, z$ and that our initial customers are a P at queue X at position i', through to a P′ at queue Y at position j'. The cooperation set between all the queues is L_Q.

$$\mathrm{QState} \stackrel{def}{=} \mathrm{QA}_{\mathrm{pop}a} \underset{L_R}{\bowtie} \mathrm{QB}_{\mathrm{pop}b} \ldots \underset{L_R}{\bowtie} \mathrm{QZ}_{\mathrm{pop}z}$$

$$\mathrm{Customers} \stackrel{def}{=} \mathrm{P}_{Xi'} \underset{L_R}{\bowtie} \ldots \mathrm{P'}_{Yj'}$$

$$\mathrm{Servers} \stackrel{def}{=} \mathrm{S_A} \underset{L_Q}{\bowtie} \mathrm{S}^{\mathrm{B}}_{\mathrm{till}}$$

Finally, we can write our system equation for the transformed model:

$$\mathrm{Sys} \stackrel{def}{=} \mathrm{QState} \underset{L_{Sys}}{\bowtie} \mathrm{Customers} \underset{L_{Sys}}{\bowtie} \mathrm{Servers}$$

6 Worked Example

Consider a very simple model of a small intranet, with separate Web servers serving two clients, as depicted in Fig. 3. Each customer fetches a page from the first server with a *getpage* action, then goes to the second server for some related resources (the *getimages* action), before returning to the first server.

The server S_{page} doles out pages and the server S_{images} serves images. After S_{page} has issued a *getpage*, it releases the client from the first queue, A, with the *send* action, which routes the client on to the next queue, B (only if there is room in the downstream buffer for that client). The S_{images} server, after sending images to the client, then does some internal processing before issuing a *repeat* action which routes the client back to the first buffer again.

Clearly there is contention for the second buffer which can fill up and block client movement from the first buffer. In the quantitative analysis below, where we generate passage-time distributions for the time from a *getpage* action to a *repeat* action, we will see that increasing the rate of the *getpage* action only has a limited effect on the overall passage.
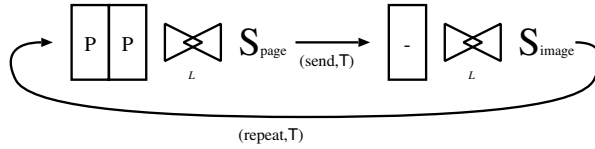
Fig. 3. Two Web servers with two clients. The cooperation set, $L = \{getpage, getimages\}$

$$\mathrm{P} \stackrel{\mathrm{def}}{=} (getpage, r_{getpage}).\mathrm{P_{ready}}$$

$$\mathrm{P_{ready}} \stackrel{\mathrm{def}}{=} (getimages, r_{getimages}).\mathrm{P}$$

$$\mathrm{S_{page}} \stackrel{\mathrm{def}}{=} (getpage, \top).(send, r_{send}).\mathrm{S_{page}}$$

$$\mathrm{S_{images}} \stackrel{\mathrm{def}}{=} (getimages, \top).(process, r_{proc}).(repeat, r_{rep}).\mathrm{S_{images}}$$

$$\mathrm{Q:A} \stackrel{\mathrm{def}}{=} (send, \top) \to \mathrm{Q:B}$$

$$\mathrm{Q:B} \stackrel{\mathrm{def}}{=} (repeat, \top) \to \mathrm{Q:A}$$

$$\mathrm{QNet:Sys} \stackrel{\mathrm{def}}{=} (\mathrm{Q:A}[\mathrm{P,P}] \underset{L}{\bowtie} \mathrm{S_{page}}) \;||\; (\mathrm{Q:B}[-] \underset{L}{\bowtie} \mathrm{S_{images}})$$
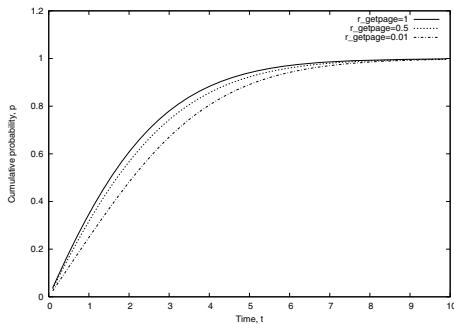
where $L = \{getpage, getimages\}$.

Fig. 4. Passage-time cumulative distribution functions from the first *getpage* action to the *repeat* action: for $r_{getpage} = 0.01, 0.5, 1.0$
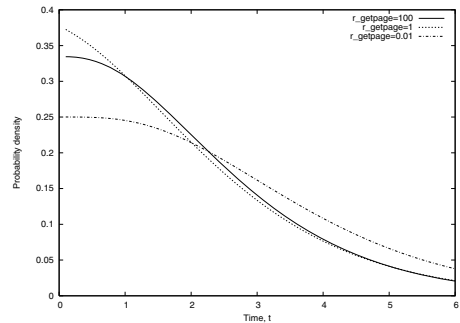
Fig. 5. Passage-time density functions from the first *getpage* action to the *repeat* action: for $r_{getpage} = 0.01, 1.0, 100$

In Fig. 4, we measure the cumulative distribution function of the passage from a *getpage* action until the next *repeat* using the ipc/DNAmaca toolset [3]. In increasing the rate of the *getpage* action, we see some benefit at first, $r_{getpage} = 0.01$ to 1.0. However, when plotting the density function of the same passage, we see that the time for the passage to complete also depends on the other rates and structures in the system. To move from queue A to B a customer must wait for an empty slot at queue B, then a *send* action to route it there. The next *repeat* action will only occur once the customer has performed the *getimages* action and can then perform a *repeat* to return to the first server. Hence in Fig. 5, we see that in looking at the passage for $r_{getpage} = 100$, we see that we don't get a monotone improvement in probability of early passage completion as we see from $r_{getpage} = 0.01$ to 1.0, but instead the extra contention caused serves to worsen the overall passage metric.

7 Conclusions and Future Work

In this paper, we have introduced PEPA Queues, a formalism for expressing individual customer behaviour and routing in queueing networks. Queueing customers are described in PEPA, as are individual servers in the network, with interactions between customers and servers deciding the exact routing pathway for a customer. This synergy of a behavioural description and a spatial formalism is similar to PEPA nets but is obviously tailored to situations where queueing models would be a more appropriate spatial formalism.

We further presented an operational semantics for PEPA Queues in terms of an underlying PEPA model and presented a worked example of a simple web server system. We finished by demonstrating passage-time analysis across the worked example.

As described, PEPA Queues are restricted to closed networks of bounded queues, in keeping with PEPA's handling of finite state spaces. It would also be desirable to handle unbounded queues, which will require further work to describe the semantics of the extended system.

The most appealing next step would be to exploit known queueing network properties directly, for certain classes of PEPA Queue. By operating on a network of PEPA Queues directly, rather than in terms of their translation into PEPA, we could exploit known results – for example, separability in queueing networks. This is one of the appealing aspects of queueing networks, which can let us escape the typical state space explosion we have with Markovian modelling in general. There are also popular queueing network features which deserve further investigation:

Unbounded buffers and open networks Handling open networks with unbounded buffers could allow us to apply many known theoretical queueing results to our models. For the translation we would need to use a formalism that allows infinite, regular state spaces, for which there are existing extensions to

PEPA [2,4].

Sources and sinks In a sense, these are just queues with unbounded buffers. However, if all our infinite queues have either no inputs or no outputs, then there is still a straightforward translation to PEPA which would follow the style of customer-centric models [5].

Inter-customer cooperation In some real-world systems, the customers may cooperate with one another in the queue. This may be a reasonable way to represent negative customers [8]. These cooperations could be just between neighbours or amongst all the customers awaiting service. Zero-automatic queues [7], for example, allow neighbouring customers of a shared group of classes to interact and coalesce into a single customer.

It would also be useful to offer translations into other formalisms where we can retain more of the inherent structure. We are investigating a translation into PEPA nets, to be followed by other suitable target systems.

We have introduced a basic syntax for defining networks of queues with embedded PEPA, together with a description of how to rewrite those models into PEPA.

References

[1] Ajmone Marsan, M., G. Conte and G. Balbo, *A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems **2** (1984), pp. 93–122.

[2] Argent-Katwala, A., "A compositional, collaborative performance pipeline," Ph.D. thesis, Imperial College, London, United Kingdom (2006).

[3] Bradley, J. T., N. J. Dingle, S. T. Gilmore and W. J. Knottenbelt, *Derivation of passage-time densities in PEPA models using ipc: the Imperial PEPA Compiler*, in: G. Kotsis, editor, *MASCOTS'03, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (2003), pp. 344–351.

[4] Clark, G., "Techniques for the Construction and Analysis of Algebraic Performance Models," Ph.D. thesis, Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK (1994).

[5] Clark, G. and J. Hillston, *Product form solution for an insensitive stochastic process algebra structure*, Performance Evaluation **50** (2002), pp. 129–151.

[6] Clark, G. and W. Sanders, *Implementing a stochastic process algebra within the Möbius modeling framework*, in: L. de Alfaro and S. Gilmore, editors, *Proc. of the 1st joint PAPM-PROBMIV Workshop*, Lecture Notes in Computer Science **2165** (2001), pp. 200–215.

[7] Dao-Thi, T.-H. and J. Mairesse, *Zero-automatic queues*, in: M. Bravetti, L. Kloul and G. Zavattaro, editors, *EPEW/WS-FM*, Lecture Notes in Computer Science **3670** (2005), pp. 64–78.

[8] E. Gelenbe, E., *Queuing networks with negative and positive customers*, Journal of Applied Probability **28** (1991), pp. 656–663.

[9] Gilmore, S. and J. Hillston, *The PEPA workbench: A tool to support a process algebra-based approach to performance modelling*, in: G. Haring and G. Kotsis, editors, *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Lecture Notes in Computer Science **794** (1994), pp. 353–368.

[10] Gilmore, S., J. Hillston and M. Ribaudo, *PEPA nets: A structured performance modelling formalism,* in: A. Field et al., editor, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation,* number 2324 in Lecture Notes in Computer Science (2002), pp. 111–130.

[11] Hillston, J., "A Compositional Approach to Performance Modelling," Distinguished Dissertations in Computer Science **12**, Cambridge University Press, 1996.

[12] Kwiatkowska, M. Z., G. Norman and D. Parker, *Prism: Probabilistic symbolic model checker,* in: A. Field et al., editor, *TOOLS'02, Proceedings of Computer Performance Evaluation: Modelling Techniques and Tools,* Lecture Notes in Computer Science **2324** (2002).

A Translation of example model

The original system from Sect. 3.1 in PEPA Queues:

$$P_{\text{eggs}} \stackrel{\text{def}}{=} (eggs, \top).P_{\text{eggs}} + (pay, \top).P_{\text{eggs}}$$

$$P_{\text{milk}} \stackrel{\text{def}}{=} (milk, \top).P_{\text{milk}} + (pay, \top).P_{\text{milk}}$$

$$S_{\text{eggs}} \stackrel{\text{def}}{=} (eggs, \top).S_{\text{noeggs}} + (expire, e_{eggs}).S_{\text{noeggs}}$$

$$S_{\text{noeggs}} \stackrel{\text{def}}{=} (restockeggs, r_{eggs}).S_{\text{eggs}}$$

$$S_{\text{milk}} \stackrel{\text{def}}{=} (milk, \top).S_{\text{nomilk}} + (expire, e_{milk}).S_{\text{nomilk}}$$

$$S_{\text{nomilk}} \stackrel{\text{def}}{=} (restockmilk, r_{milk}).S_{\text{milk}}$$

$$S_{\text{wait}} \stackrel{\text{def}}{=} (eggs, \top).S_{\text{sending}} + (milk, \top).S_{\text{sending}}$$

$$S_{\text{sending}} \stackrel{\text{def}}{=} (send, \frac{9r}{10}).S_{\text{wait}} + (redo, \frac{r}{10}).S_{\text{wait}}$$

$$S_{\text{serve}} \stackrel{\text{def}}{=} (S_{\text{eggs}} \parallel S_{\text{milk}}) \underset{\{milk, eggs\}}{\bowtie} S_{\text{wait}}$$

$$S_{\text{till}} \stackrel{\text{def}}{=} (pay, \top).(process, p).(repeat, s).S_{\text{till}}$$

$$Q{:}A \stackrel{\text{def}}{=} (redo, \top) \rightarrow Q{:}A + (send, \top) \rightarrow Q{:}B$$

$$Q{:}B \stackrel{\text{def}}{=} (repeat, \top) \rightarrow Q{:}A$$

$$QNet{:}Sys \stackrel{\text{def}}{=} (Q{:}A[P_{\text{milk}}, P_{\text{eggs}}, P_{\text{eggs}}, P_{\text{milk}}] \underset{\{eggs, milk, pay\}}{\bowtie} S_{\text{serve}})$$

$$\parallel (Q{:}B[-, -] \underset{\{eggs, milk, pay\}}{\bowtie} S_{\text{till}})$$

Our flat PEPA model uses two cooperation sets:

$$L_R = \{repeat_{A0}, repeat_{A1}, repeat_{A2}, redo_{A0}, redo_{A1}, redo_{A2}, send_{B0}, send_{B1}\}$$

$$L_{Sys} = L_R \cup \{eggs_A, milk_A, pay_A, eggs_B, milk_B, pay_B\}$$

The population trackers for the two queues:

$$QA_{\text{pop0}} \stackrel{def}{=} (repeat_{A0}, \top).QA_{\text{pop1}}$$

$$QA_{\text{pop1}} \stackrel{def}{=} (repeat_{A1}, \top).QA_{\text{pop2}} + (redo_{A1}, \top).QA_{\text{pop1}}$$

$$+ (send_{B0}, \top).QA_{\text{pop0}} + (send_{B1}, \top).QA_{\text{pop0}}$$

$$QA_{pop2} \stackrel{def}{=} (repeat_{A2}, \top).QA_{pop3} + (redo_{A2}, \top).QA_{pop2}$$
$$+ (send_{B0}, \top).QA_{pop1} + (send_{B1}, \top).QA_{pop1}$$
$$QA_{pop3} \stackrel{def}{=} (repeat_{A3}, \top).QA_{pop4} + (redo_{A3}, \top).QA_{pop3}$$
$$+ (send_{B0}, \top).QA_{pop2} + (send_{B1}, \top).QA_{pop2}$$
$$QA_{pop4} \stackrel{def}{=} (send_{B0}, \top).QA_{pop3} + (send_{B1}, \top).QA_{pop3}$$
$$QB_{pop0} \stackrel{def}{=} (send_{B0}, \top).QB_{pop1}$$
$$QB_{pop1} \stackrel{def}{=} (send_{B1}, \top).QB_{pop2}$$
$$+ (repeat_{A0}, \top).QB_{pop0} + (repeat_{A1}, \top).QB_{pop0}$$
$$+ (repeat_{A2}, \top).QB_{pop0} + (repeat_{A3}, \top).QB_{pop0}$$
$$QB_{pop2} \stackrel{def}{=} (repeat_{A0}, \top).QB_{pop1} + (repeat_{A1}, \top).QB_{pop1}$$
$$+ (repeat_{A2}, \top).QB_{pop1} + (repeat_{A3}, \top).QB_{pop1}$$

The server components:

$$S^A_{eggs} \stackrel{def}{=} (eggs_A, \top).S^A_{noeggs} + (expire, e_{eggs}).S^A_{noeggs}$$
$$S^A_{noeggs} \stackrel{def}{=} (restockeggs, r_{eggs}).S^A_{eggs}$$

$$S^A_{milk} \stackrel{def}{=} (milk_A, \top).S^A_{nomilk} + (expire, e_{milk}).S^A_{nomilk}$$
$$S^A_{nomilk} \stackrel{def}{=} (restockmilk, r_{milk}).S^A_{milk}$$

$$S^A_{wait} \stackrel{def}{=} (eggs_A, \top).S^A_{sending} + (milk_A, \top).S^A_{sending}$$
$$S^A_{sending} \stackrel{def}{=} (send_{B0}, \frac{9r}{10}).S^A_{wait} + (send_{B1}, \frac{9r}{10}).S^A_{wait}$$
$$+ (redo_{A0}, \frac{r}{10}).S^A_{wait} + (redo_{A1}, \frac{r}{10}).S^A_{wait}$$
$$+ (redo_{A2}, \frac{r}{10}).S^A_{wait} + (redo_{A3}, \frac{r}{10}).S^A_{wait}$$

$$S^A_{serve} \stackrel{def}{=} (S^A_{eggs} \parallel S^A_{milk}) \underset{\{milk_A, eggs_A\}}{\bowtie} S^A_{wait}$$

$$S^B_{till} \stackrel{def}{=} (pay_B, \top).S^B_{till0}$$
$$S^B_{till0} \stackrel{def}{=} (process, p).S^B_{till1}$$
$$S^B_{till1} \stackrel{def}{=} (repeat_{A0}, s).S^B_{till} + (repeat_{A1}, s).S^B_{till} +$$
$$(repeat_{A2}, s).S^B_{till} + (repeat_{A3}, s).S^B_{till}$$

So the initial state, from the queueing network system equation, is:

$$QState \stackrel{def}{=} QA_{pop3} \underset{L_R}{\bowtie} QB_{pop0}$$

$$\text{Servers} \stackrel{\text{def}}{=} S_{\text{serve}}^A \parallel S_{\text{till}}^B$$

Note that each occurrence of a routing action is expanded into a choice of all the variants of that action, and that we have named the anonymous sub-components of S_{till} to facilitate this.

The customer components:

$$P_{\text{eggs}}^{A1} \stackrel{\text{def}}{=} (eggs_A, r_e).P_{\text{eggs}}^{A1} + (pay_A, p_e).P_{\text{eggs}}^{A1} + (send_{B0}, \top).P_{\text{eggs}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{eggs}}^{A2} + (redo_{A2}, \top).P_{\text{eggs}}^{A3} + (redo_{A3}, \top).P_{\text{eggs}}^{A4}$$

$$P_{\text{eggs}}^{A2} \stackrel{\text{def}}{=} (eggs_A, r_e).P_{\text{eggs}}^{A2} + (pay_A, p_e).P_{\text{eggs}}^{A2} + (send_{B0}, \top).P_{\text{eggs}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{eggs}}^{A2} + (redo_{A2}, \top).P_{\text{eggs}}^{A3} + (redo_{A3}, \top).P_{\text{eggs}}^{A4}$$

$$P_{\text{eggs}}^{A3} \stackrel{\text{def}}{=} (eggs_A, r_e).P_{\text{eggs}}^{A3} + (pay_A, p_e).P_{\text{eggs}}^{A3} + (send_{B0}, \top).P_{\text{eggs}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{eggs}}^{A2} + (redo_{A2}, \top).P_{\text{eggs}}^{A3} + (redo_{A3}, \top).P_{\text{eggs}}^{A4}$$

$$P_{\text{eggs}}^{A4} \stackrel{\text{def}}{=} (eggs_A, r_e).P_{\text{eggs}}^{A4} + (pay_A, p_e).P_{\text{eggs}}^{A4} + (send_{B0}, \top).P_{\text{eggs}}^{B1} +$$
$$(redo_{A1}, \top).P_{\text{eggs}}^{A2} + (redo_{A2}, \top).P_{\text{eggs}}^{A3} + (redo_{A3}, \top).P_{\text{eggs}}^{A4}$$

$$P_{\text{eggs}}^{B1} \stackrel{\text{def}}{=} (eggs_B, r_e).P_{\text{eggs}}^{B1} + (pay_B, p_e).P_{\text{eggs}}^{B1} + (repeat_{A0}, \top).P_{\text{eggs}}^{A1}$$
$$+ (repeat_{A1}, \top).P_{\text{eggs}}^{A2} + (repeat_{A2}, \top).P_{\text{eggs}}^{A3} + (repeat_{A3}, \top).P_{\text{eggs}}^{A4}$$

$$P_{\text{eggs}}^{B2} \stackrel{\text{def}}{=} (eggs_B, r_e).P_{\text{eggs}}^{B2} + (pay_B, p_e).P_{\text{eggs}}^{B2} + (repeat_{A0}, \top).P_{\text{eggs}}^{A1}$$
$$+ (repeat_{A1}, \top).P_{\text{eggs}}^{A2} + (repeat_{A2}, \top).P_{\text{eggs}}^{A3} + (repeat_{A3}, \top).P_{\text{eggs}}^{A4}$$

$$P_{\text{milk}}^{A1} \stackrel{\text{def}}{=} (milk_A, r_m).P_{\text{milk}}^{A1} + (pay_A, p_m).P_{\text{milk}}^{A1} + (send_{B0}, \top).P_{\text{milk}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{milk}}^{A2} + (redo_{A2}, \top).P_{\text{milk}}^{A3} + (redo_{A3}, \top).P_{\text{milk}}^{A4}$$

$$P_{\text{milk}}^{A2} \stackrel{\text{def}}{=} (milk_A, r_m).P_{\text{milk}}^{A2} + (pay_A, p_m).P_{\text{milk}}^{A2} + (send_{B0}, \top).P_{\text{milk}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{milk}}^{A2} + (redo_{A2}, \top).P_{\text{milk}}^{A3} + (redo_{A3}, \top).P_{\text{milk}}^{A4}$$

$$P_{\text{milk}}^{A3} \stackrel{\text{def}}{=} (milk_A, r_m).P_{\text{milk}}^{A3} + (pay_A, p_m).P_{\text{milk}}^{A3} + (send_{B0}, \top).P_{\text{milk}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{milk}}^{A2} + (redo_{A2}, \top).P_{\text{milk}}^{A3} + (redo_{A3}, \top).P_{\text{milk}}^{A4}$$

$$P_{\text{milk}}^{A4} \stackrel{\text{def}}{=} (milk_A, r_m).P_{\text{milk}}^{A4} + (pay_A, p_m).P_{\text{milk}}^{A4} + (send_{B0}, \top).P_{\text{milk}}^{B1}$$
$$+ (redo_{A1}, \top).P_{\text{milk}}^{A2} + (redo_{A2}, \top).P_{\text{milk}}^{A3} + (redo_{A3}, \top).P_{\text{milk}}^{A4}$$

$$P_{\text{milk}}^{B1} \stackrel{\text{def}}{=} (milk_A, r_m).P_{\text{milk}}^{B1} + (pay_A, p_m).P_{\text{milk}}^{B1} + (repeat_{A0}, \top).P_{\text{milk}}^{A1}$$
$$+ (repeat_{A1}, \top).P_{\text{milk}}^{A2} + (repeat_{A2}, \top).P_{\text{milk}}^{A3} + (repeat_{A3}, \top).P_{\text{milk}}^{A4}$$

$$P_{\text{milk}}^{B2} \stackrel{\text{def}}{=} (milk_A, r_m).P_{\text{milk}}^{B2} + (pay_A, p_m).P_{\text{milk}}^{B2} + (repeat_{A0}, \top).P_{\text{milk}}^{A1}$$
$$+ (repeat_{A1}, \top).P_{\text{milk}}^{A2} + (repeat_{A2}, \top).P_{\text{milk}}^{A3} + (repeat_{A3}, \top).P_{\text{milk}}^{A4}$$

Again, using we derive the initial state of the customers from the system equation of the PEPA Queues model:

$$\text{Customers} \stackrel{\text{def}}{=} P_{\text{milk}}^{A4} \underset{L_{Sys}}{\bowtie} P_{\text{eggs}}^{A3} \underset{L_{Sys}}{\bowtie} P_{\text{eggs}}^{A2} \underset{L_{Sys}}{\bowtie} P_{\text{milk}}^{A1}$$

So the system equation for the transformed model is:

$$\text{Sys} \stackrel{\text{def}}{=} \text{QState} \underset{L_{Sys}}{\bowtie} \text{Customers} \underset{L_{Sys}}{\bowtie} \text{Servers}$$