PEPA nets: A structured performance modelling formalism

Stephen Gilmore[1], Jane Hillston[1], and Marina Ribaudo[2]

[1] Laboratory for Foundations of Computer Science, The University of Edinburgh, Edinburgh EH9 3JZ, Scotland. Email: {stg, jeh}@dcs.ed.ac.uk

[2] Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Via Dodecaneso 35, 16146 Genova, Italia. Email: ribaudo@disi.unige.it

Abstract. In this paper we describe a formalism which uses the stochastic process algebra PEPA as the inscription language for labelled stochastic Petri nets. Viewed in another way, the net is used to provide a structure for combining related PEPA systems. The combined modelling language naturally represents such applications as mobile code systems where the PEPA terms are used to model the program code which moves between network hosts (the places in the net). We describe the implementation of a tool to support this modelling formalism and apply this to model a peer-to-peer filestore.

1 Introduction

Variants of Petri nets have been widely used in the description and performance analysis of computer, telecommunications and manufacturing systems [1]. The appeal of Petri nets as a modelling formalism is easy to see. They provide a graphical presentation of a model which has an easily accessible interpretation and they also have the advantage of being supported by an unambiguous formal interpretation.

In their use as performance modelling languages stochastic Petri nets have recently been joined by stochastic process algebras such as PEPA [2], EMPA [3] and IMC [4]. Stochastic process algebras lack the attractive graphical presentation of Petri nets and properties such as the depiction of causality and conflict in a model. In contrast though, in stochastic process algebras an explicit compositional structure is imposed on the model. This structure makes the model easy to understand, may alleviate problems of model construction and can be exploited for both qualitative and quantitative analysis. A comparison of these two modelling formalisms [5] concludes that "there is scope for future work incorporating the attractive characteristics of the formalisms, such as structural analysis or functional abstraction, from one paradigm into the other". Some work has been done in this area in beginning to develop a structural theory for process algebras [6] on the one hand and in importing composition operations from stochastic process algebras into Well-formed nets on the other [7]. The present work considers using both Petri nets and process algebras together as a single, structured performance modelling formalism. There is some reason to believe that these two formalisms would complement each other. A recent paper [8] gives an example of a system which can be modelled more easily in one formalism than the other.

Petri nets have previously been combined with other modelling formalisms such as the lazy functional programming language Haskell (used with non-stochastic Petri nets in [9]) and queueing models (used with generalised stochastic Petri nets in [10]). The combination of stochastic Petri nets with queueing networks in particular has been a source of inspiration to several authors. Earlier work in this area includes Bause's *Queueing Petri nets* [11] and Haverkort's *Dynamic Queueing Networks* [12]. An extension of (non-stochastic) Petri nets which provides modelling concepts similar to ours is Valk's *Elementary Object systems* [13]. The tokens in an elementary object system are themselves Petri nets having individual dynamic behaviour.

Coloured Petri nets are a high-level form of classical Petri nets. The plain (indistinguishable) tokens of a classical Petri net are replaced by arbitrary terms which are distinguishable. In stochastic Petri nets the transitions from one marking to another are associated with a random variable drawn from an exponential distribution. Here we consider coloured stochastic Petri nets where the colours used as the tokens of the net are PEPA components. We refer to these as *PEPA nets* from here on.

Section 2 introduces the notation and terminology of PEPA nets, to give the reader an informal explanation of the ideas. However, PEPA is a formal language with a precise semantic definition and so in Section 3 we present the operational semantics of PEPA nets. In Section 4 we present a case study of a simple mobile agent system modelled as a PEPA net. Having presented the reader an example of modelling with PEPA nets we then compare them to the related modelling formalisms of Petri nets and the PEPA stochastic process algebra in Section 5. In each case we seek to show that PEPA nets offer some expressivity which is not directly offered by the other formalisms. In Section 6 we discuss tool support for this formalism. Section 7 is a more detailed case study. Further work is listed in Section 8. Concluding remarks are presented in Section 9.

2 PEPA nets

In this section we present the concepts and definitions used in PEPA nets. We assume that the reader is familiar with the basic concepts of process algebras and Petri nets. Readers who are unfamiliar with the PEPA process algebra are referred to Appendix A for an introduction.

There are two types of change of state in a PEPA net. We refer to these as *firings* of the net and as *transitions* of PEPA components. The intention behind having two types of change of state is that we can use these to model changes which take place on different scales. Transitions of PEPA components will typically be used to model small-scale (or *local*) changes of state as components undertake activities. Firings of the net will typically be used to model macro-step (or *global*) changes of state such as context switches, breakdowns and repairs, one thread yielding to another, or a mobile software agent moving from one network host to another. We will return to a mobile agent example later but here we wish just to point out that our motivation is primarily to model systems which have two levels of change of state and not to develop a stochastic process algebra for specifically modelling mobile code applications. A suitable formalism already exists for the latter purpose, Priami's *stochastic π-calculus* [14].

A firing in a PEPA net causes the transfer of one token from one place to another. The token which is moved is a PEPA component, which causes a change in the remainder of the evaluation both in the source (where existing co-operations with other components now can no longer take place) and in the target (where previously disabled co-operations are now enabled by the arrival of an incoming component which can participate in these interactions). Firings have global effect because they involve components at more than one place in the net.

A transition in a PEPA net takes place whenever a transition of a PEPA component can occur (either individually, or in co-operation with another component). Transitions can only take place between components which are resident in the same place in the net. The PEPA net formalism does not allow components at different places in the net to co-operate on a shared activity. An analogy is with message-passing distributed systems without shared-memory where software components on the same host can exchange information without incurring a communication overhead but software components on different hosts cannot. Transitions in a PEPA net have local effect because they involve only components at one place in the net.

A PEPA net is made up of PEPA *contexts*, one at each place in the net. The notion of context might seem an unusual one to have to employ here but contexts have previously proved useful in definitions of classical process algebras [15], and in the PEPA stochastic process algebra [16]. Contexts contain *cells*. Like a memory location in an imperative program, a cell is a storage area to be filled by a datum of a particular type. In particular in a PEPA net, a cell is a storage area dedicated to storing a PEPA component. The components which fill cells can circulate as the tokens of the net. Components which are not in a designated cell are static and cannot move. Static components which cannot move are not included in most versions of Petri nets; the closest commonly-known idea is that of "self loops" where a token is deleted from a place and then immediately added again. They prove useful here because our concept of token is more complex than most. Static components act as cooperation partners in synchronisation activities with tokens.

We use the notation $P[\_]$ to denote a context which could be filled by the PEPA component P or one with the same alphabet. If P has derivatives P' and P'' only and no other component has the same alphabet as P then there are four possible values for such a context: $P[\_]$, $P[P]$, $P[P']$ and $P[P'']$. $P[\_]$ enables no transitions. $P[P]$ enables the same transitions as P. $P[P']$ enables the same transitions as P'. $P[P'']$ enables the same transitions as P''.

2.1 Markings in a PEPA net

The *marking* of a classical Petri net records the number of tokens which are resident at each place in the net. Since the tokens of a classical Petri net are indistinguishable it is sufficient only to record their number and one could present the marking of a Petri net with places P_1, P_2 and P_3 as shown below.

$$\begin{array}{l} P_1 : 2 \\ P_2 : 1 \\ P_3 : 0 \end{array}$$

If an ordering is imposed on the places of the net a more compact representation of the marking can be used. Place names are omitted and the marking can be written using vector notation thus, $(2, 1, 0)$.

Consider now a PEPA net with places P_1, P_2 and P_3 as shown below.

$$P_1[p] \stackrel{\mathit{def}}{=} P[p] \underset{L}{\bowtie} Q$$

$$P_2[p] \stackrel{\mathit{def}}{=} P[p] \underset{K}{\bowtie} R$$

$$P_3[p] \stackrel{\mathit{def}}{=} P[p] \underset{K \cup L}{\bowtie} (Q \parallel R)$$

From their uses in the contexts at each place we see that P is a component which can move as a token around the net whereas Q and R are static components which cannot move. There is a copy of Q at place P_1 and another at P_3. There is a copy of R at place P_2 and another at P_3.

Given the above definitions for the places in this PEPA net, we can denote a marking of this net by $(P_1[P], P_2[\_], P_3[\_])$. In general, a context may have more than one parameter, to be filled by PEPA components of different types, so the vector of lists notation which we have used here is needed in general. Where an ordering is imposed on places and each context has only a single cell to be filled we can abbreviate such a marking by $(P, \_, \_)$.

2.2 Net-level transitions in a PEPA net

Transitions at the net-level of a PEPA net are labelled in a similar way to the labelled multi-transition system which records the unfolding of the state space of a PEPA model. A labelling function ℓ maps transition names into pairs of names such as (α, r) where it is possible that $\ell(t_i) = \ell(t_j)$ but $t_i \neq t_j$. The first element of a pair (α, r) specifies an *activity* which must be performed in order for a component to move from the input place of the transition to the output place. The activity type records formally the activity which must be performed if the transition is to fire. The second element is an exponentially-distributed random variable which quantifies the *rate* at which the activity can progress in conjuction with the component which is performing it.

As an example, suppose that Q is a component which is currently at place P_1 and that it can perform an activity α with rate r_1 to produce the derivative Q'.

Further, say that the net has a transition between P_1 and P_2 labelled by (α, r_2). If Q performs activity α in this setting it will be removed from P_1 (leaving behind an empty cell) and deposited into P_2 (filling an empty cell there).

3 Semantics

The PEPA language is formally defined by a small-step operational semantics as used in the definition of Milner's CCS and other process algebras. In order to describe the firing rule for PEPA nets formally we need a relational operator which is to be used to express the fact that there exists a particular transition in the net superstructure. This operator must have the properties that it identifies the source and target of the transition and that it records the activity which is to be performed in order for a component to cross this transition, moving from the source to the target. We use the notation

$$P_1 \overset{(\alpha, r)}{\longrightarrow\!\!\Box\!\!\longrightarrow} P_2$$

to capture the information that there is a transition connecting place P_1 to place P_2 labelled by (α, r). This relation captures static information about the structure of the net, not dynamic information about its behaviour. We could describe the net structure in a PEPA net using a list of such assertions but the more familiar graphical presentation of a net presents the same information in a more accessible way.

When we wish to express the fact that no such labelled transition exists connecting place P_1 to place P_2, we draw a stroke through the box in the middle of the arrow symbol. This relation can be defined in terms of the relation above.

$$P_1 \overset{(\alpha, r)}{\longrightarrow\!\!\not\Box\!\!\longrightarrow} P_2 \;=\; \nexists P_2 \cdot P_1 \overset{(\alpha, r)}{\longrightarrow\!\!\Box\!\!\longrightarrow} P_2$$

The introduction of contexts requires an extension to the syntax of PEPA. This extension is presented in Figure 1.

The semantic rules for PEPA nets are provided in Figure 2. The Cell rule conservatively extends the PEPA semantics to define that a cell which is filled by a component P has the same transitions as P itself. A healthiness condition on the rule (also called a *typing judgement*) requires a context such as $P[\_]$ to be filled with a component which has the same alphabet as P. We write $P =_a P'$ to state that P and P' have the same alphabet. There are no rules to infer transitions for an empty cell because an empty cell enables no transitions.

The Transition rule states that the net has local transitions which change only a single component in the marking vector. This rule also states that these transitions agree with the transitions which are generated by the PEPA semantics (including the extension for contexts). The second premise of the Transition rule mandates that a local transition α can only occur in a place in the net which does not have an outgoing arc labelled by α. Note that this negative requirement is a static requirement related to the structure of the net, not the negation of

$$
\begin{array}{lrlrlr}
N ::= D^{+}M & \text{(net)} & M ::= (P, \ldots) & \text{(marking)} & D ::= I \overset{def}{=} S & \text{(defn)} \\
\multicolumn{2}{c}{\text{(definitions and marking)}} & \multicolumn{2}{c}{\text{(a vector of components)}} & \multicolumn{2}{c}{\text{(identifier declaration)}} \\
\\
S ::= (\alpha, r).S & \text{(prefix)} & P ::= P \underset{L}{\bowtie} P & \text{(cooperation)} & C ::= \text{‘\_’} & \text{(empty)} \\
\quad \mid \ S + S & \text{(choice)} & \quad \mid \ P/L & \text{(hiding)} & \quad \mid \ P & \text{(full)} \\
\quad \mid \ I & \text{(identifier)} & \quad \mid \ P[C] & \text{(cell)} & & \\
 & & \quad \mid \ I & \text{(identifier)} & & \\
\multicolumn{2}{c}{\text{(sequential components)}} & \multicolumn{2}{c}{\text{(concurrent components)}} & \multicolumn{2}{c}{\text{(cell term expressionss)}}
\end{array}
$$

Fig. 1. The syntax of PEPA extended with contexts

the transition relation which is being defined. Thus, the rule cannot fail to be *stratifiable* [17].

The Firing rule takes one marking of the net to another marking by performing a PEPA activity and moving a PEPA component from the input place to the output place. This has the effect that two entries in the marking vector change simultaneously. The rate at which the activity is performed is calculated as in the PEPA semantics of co-operation. That is, the rate is adjusted to reflect the inability of the slower co-operand to function at the rate of the faster co-operand, using the well-known notion of *apparent rate*. The definition of apparent rate can be found in [2].

Cell:

$$
\frac{P' \xrightarrow{(\alpha, r)} P''}{P[P'] \xrightarrow{(\alpha, r)} P[P'']} \quad (P =_a P')
$$

Transition:

$$
\frac{P \xrightarrow{(\alpha, r)} P' \qquad P \overset{(\alpha, r')}{\longrightarrow\!\!\!\!\!\!\!\!\not\square\;\;\;} Q}{(\ldots, P, \ldots) \xrightarrow{(\alpha, r)} (\ldots, P', \ldots)}
$$

Firing:

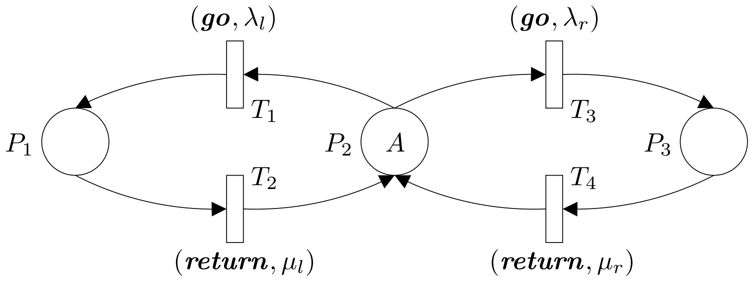
$$
\frac{Q \xrightarrow{(\alpha, r_1)} Q' \qquad P_i \overset{(\alpha, r_2)}{\longrightarrow\!\!\!\!\!\!\!\!\square\;\;\;} P_j}{(\ldots, P_i[Q], \ldots, P_j[\_], \ldots) \xrightarrow{(\alpha, R)} (\ldots, P_i[\_], \ldots, P_j[Q'], \ldots)}
$$

Fig. 2. Semantics of PEPA net firings

4 Example: a mobile agent system

We present a small example to reinforce the reader's understanding of PEPA nets. In this example a roving agent visits three sites. It interacts with static software components at these sites and has two kinds of interactions. When visiting a site where a network probe is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic. When it returns to the central co-ordinating site it dumps the data which it has harvested to the master probe. The master probe performs a computationally expensive statistical analysis of the data. The structure of the system allows this computation to be overlapped with the agent's communication and data gathering. The marshalling and unmarshalling costs for mobile code applications are a significant expense so overlapping this with data processing allows some of this expense to be offset.

The structure of the application is as represented by the PEPA net in Figure 3. This marking of the net shows the mobile agent resident at the central co-ordinating site. As a mnemonic, in both the net and the PEPA description, we print in bold the names of those activities which can cause a firing of the net. In this example, those activities are ***go*** and ***return***.



$(\boldsymbol{go}, \lambda_l)$
$(\boldsymbol{go}, \lambda_r)$
T_1
T_3
P_1
P_2
A
P_3
T_2
T_4
$(\boldsymbol{return}, \mu_l)$
$(\boldsymbol{return}, \mu_r)$

Fig. 3. A simple mobile agent system

Formally, we define the places of the net as shown in the PEPA context definitions below. P_2 has a local state denoted by P_2'.

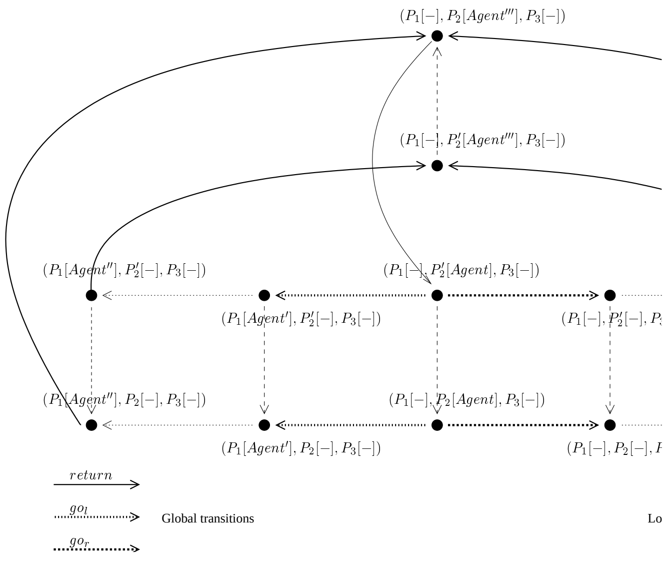
$$P_1[A] \overset{def}{=} Agent[A] \underset{\{interrogate\}}{\bowtie} Probe \qquad P_3[A] \overset{def}{=} Agent[A] \underset{\{interrogate\}}{\bowtie} Probe$$

$$P_2[A] \overset{def}{=} Agent[A] \underset{\{dump\}}{\bowtie} Master \qquad P_2'[A] \overset{def}{=} Agent[A] \underset{\{dump\}}{\bowtie} Master'$$

The initial marking of the net is $(\_, Agent, \_)$. The behaviour of the components is given by the following PEPA definitions.

$$\begin{aligned} Agent &\overset{def}{=} (\boldsymbol{go}, \lambda).Agent' & Master &\overset{def}{=} (dump, \top).Master' \\ Agent' &\overset{def}{=} (interrogate, r_i).Agent'' & Master' &\overset{def}{=} (analyse, r_a).Master \\ Agent'' &\overset{def}{=} (\boldsymbol{return}, \mu).Agent''' & Probe &\overset{def}{=} (monitor, r_m).Probe + \\ Agent''' &\overset{def}{=} (dump, r_d).Agent & &\quad\ (interrogate, \top).Probe \end{aligned}$$

The transition system underlying this model is shown in Figure 4. The derivation of such a transition system is the first step in the performance analysis of such a system. The transition system contains the specification of a Continuous-Time Markov Chain model of the system. This CTMC is solved for its stationary distribution and performance measures are calculated from that. For a model as simple as this one we can solve it simply with Gaussian elimination. We also have available solvers such as an efficient implementation of the preconditioned biconjugate gradient method.



$(P_1[-], P_2[Agent'''], P_3[-])$
$(P_1[-], P'_2[Agent'''], P_3[-])$
$(P_1[Agent''], P'_2[-], P_3[-])$
$(P_1[-], P'_2[Agent], P_3[-])$
$(P_1[Agent'], P'_2[-], P_3[-])$
$(P_1[-], P'_2[-], P$
$(P_1[Agent''], P_2[-], P_3[-])$
$(P_1[-], P_2[Agent], P_3[-])$
$(P_1[Agent'], P_2[-], P_3[-])$
$(P_1[-], P_2[-], P$
$return$
go_l
Global transitions
Lo
go_r

Fig. 4. The transition system of the mobile agent example

5 Relating PEPA nets to other modelling formalisms

If they were to be viewed purely formally as high-level description languages for specifying continuous-time Markov chains, then PEPA nets, stochastic Petri nets and the PEPA stochastic process algebra would be considered to be equally expressive. That is to say, for a given CTMC C, it is possible to construct a high-level model in each of these three formalisms such that the underlying CTMC derived from the model is isomorphic to C.

In practice, the three languages present different sets of conceptual tools to the modeller. From the pragmatic perspective of a performance modeller who wishes to reliably encode a high-level model of a particular system then there might be reasons to select one of the languages instead of the others for this particular modelling study. In the remainder of this section we compare modelling with PEPA nets with modelling with Petri nets and the PEPA stochastic process algebra.

5.1 Relating PEPA nets to Petri nets

To illustrate the difference between PEPA nets and Petri nets we first show how to represent an ordinary k-safe stochastic Petri net as a PEPA net. In a classical stochastic Petri net tokens are indistinguishable. We can replicate this in a PEPA net by having only a single class of tokens which have only one (PEPA) state. The definition of such a token would also need to always permit firings of the net to take place. We define these tokens by summing over all of the transition activity names, for all of the transitions of the net ($tn_i \in T$).

$$\mathit{Token} \overset{\mathit{def}}{=} \sum_{tn_i \in T} (tn_i, \top).\mathit{Token}$$

To define a k-safe stochastic Petri net with a PEPA net we then need simply to specify the places of the net as being capable of storing up to k of these tokens, and making no use of static components.

$$P_i[tk_1, \ldots, tk_k] \overset{\mathit{def}}{=} \mathit{Token}[tk_1] \parallel \cdots \parallel \mathit{Token}[tk_k]$$

This reconstruction of ordinary k-safe stochastic Petri nets from PEPA nets points to the difference between the two formalisms. A PEPA net can be viewed as a Petri net where the tokens are *programmable*. The tokens of a PEPA net have state, can count, can observe activities, and can even refuse to be fired from the place where they reside. We believe that this gives the PEPA net modeller a novel conceptual modelling tool which can be used to express natural descriptions of systems with active, stateful mobile agents.

5.2 Relating PEPA nets to PEPA

The relationship between PEPA nets and PEPA is straightforward. A PEPA net with only one place and no transitions is simply a PEPA stochastic process algebra model. To explain how a PEPA net can offer added expressive power we consider a PEPA net with more than one place, such as that shown in Figure 5.

In this model a token of type P moves between place P_1 and place P_2. In doing so it decouples itself from a static component Q, located at P_1. The token thereby moves out of the scope of the cooperation set L. Cooperation sets are used to configure copies of components, coupling them to communication partners. In this way they restrict the behaviour of a component, requiring it to perform some activities (those in the cooperation set) only if they have a partner who is able to cooperate in performing them. In the example in Figure 5 if Q is unwilling to perform some of these activities then the behaviour of P will be restricted. Even if Q is willing to perform all of the activities in the cooperation

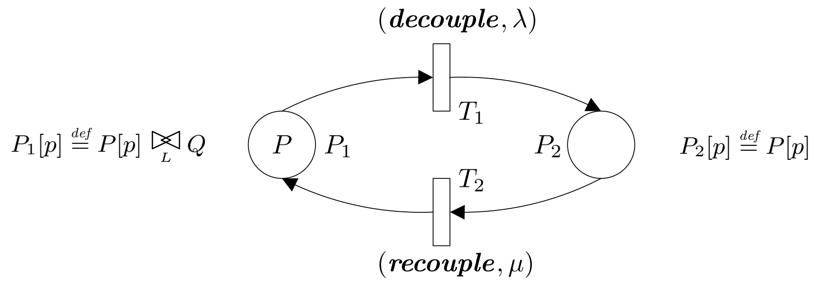

$(\boldsymbol{decouple}, \lambda)$
T_1
$P_1[p] \overset{def}{=} P[p] \underset{L}{\bowtie} Q$
P
P_1
P_2
$P_2[p] \overset{def}{=} P[p]$
T_2
$(\boldsymbol{recouple}, \mu)$

Fig. 5. Tokens in a PEPA net can decouple from a static component

set L then it can still influence the rate at which they are performed. In contrast when the token P is resident in P_2 then it is subject to no such restriction and can perform all of its activities at the rates which it itself specifies.

This concept cannot be expressed in a PEPA stochastic process algebra model. The cooperation sets used in a PEPA model impose a static communication topology on the model. In contrast a PEPA net has dynamically varying communication structure and, in consequence, a given action in a component might sometimes be performed in isolation and sometimes be performed in cooperation. The ability to express the concept of dynamically varying communication structure offers an additional conceptual tool to the performance modeller which is not available when modelling in PEPA. In this way PEPA nets strictly extend the expressiveness of the PEPA stochastic process algebra.

6 Implementation

The PEPA stochastic process algebra is supported by a range of tools including the PEPA Workbench [18] and the Möbius Modelling Framework [19]. In this section we explain the use of PEPA nets with these tools. In the first case we have extended the tool to directly support the extended formalism. In the second case we explain how to translate PEPA nets into the extension of PEPA supported by Möbius, PEPA$_k$.

6.1 The PEPA Workbench for PEPA nets

We have implemented the PEPA nets formalism as an extension of the PEPA Workbench, an existing modelling tool for PEPA. The PEPA Workbench exists in two distinct versions. The first version is an experimental research tool which is coded in the functional programming language Standard ML [20]. The second is a re-implementation of this in the Java programming language. These are known as "the ML edition" and "the Java edition" respectively.

Standard ML and Java have very different strengths. For a visual language such as the notation of Petri nets the Java language's visualisation capabilities

```
PEPA Workbench for PEPA nets Version 0.72.2 "Cramond" [14-08-2001]
Filename: agent.pepa
Compiling the model
Generating the derivation graph
The model has 12 states
The model has 35 transitions
The model has 8 firings
Writing the hash table file to agent.hash
Exiting PEPA Workbench.
```

Fig. 6. The PEPA Workbench for PEPA nets processing the mobile agent model

would suit the task much better than Standard ML. Further, there are existing Java tools for Petri nets which could be extended to provide an implementation of PEPA nets. After initial experimentation with the Standard ML version of the PEPA Workbench for PEPA nets, a graphical presentation of PEPA nets could be incorporated into the Java version of the Workbench. This implementation plan is ongoing, but at an early stage.

The Standard ML language is well suited to implementing symbolic processing applications and provides built-in support for describing datatypes such as those needed to present the abstract syntax of a formal language such as PEPA. These features made it possible to rapidly adapt the routines for generating PEPA derivation graphs to generate derivation graphs for PEPA nets. The compact transition rules presented in Figure 2 look simple on the page but they proved to be a challege to implement efficiently. Here again, the higher-order features of the Standard ML programming language proved to be useful in allowing us to form function closures from higher-order functions which fixed some of their formal parameters. This allowed us to unroll the derivation graph for the PEPA nets model without suffering a performance penalty due to accumulated parameter information.

The use of the PEPA Workbench for PEPA nets is illustrated in Figure 6. Comparing the results with the picture of the labelled transition system in Figure 4 we see that the number of states and firings agree. Two self-loops are omitted from every state in the diagram in order to avoid clutter. This accounts for the additional twenty-four transitions reported by the PEPA Workbench for PEPA nets.

The input language to the tool is an extension of the concrete syntax used for storing PEPA language models. The topology of the net is specified by providing a textual description of the places and the arcs connecting them. Providing a visual editor for PEPA nets remains as future work.

6.2 Using PEPA nets in the Möbius Modelling Framework

Möbius is a multi-formalism performance and dependability modelling tool. The modeller can specify atomic models in a number of component-based modelling formalisms, including stochastic activity networks (SANs), stochastic Petri nets (SPNs), Markov processes specified in "bucket and ball" style, queueing networks and the PEPA stochastic process algebra [21]. These atomic models are configured and composed using a graphical notation to make a structured high-level model which is composed from atomic submodels. The ability to interconnect models expressed in different formalisms is achieved through the use of an *abstract functional interface* to the models. Models can be solved either by discrete event simulation or by state-based, analytical and numerical techniques. The Möbius solvers use efficient implementations of classical sparse matrix solution methods.

Möbius implements an extension of PEPA called PEPA$_k$. It is shown in [21] that PEPA$_k$ models can be mapped to PEPA models. Here we explain how to map PEPA nets to PEPA$_k$ models.

PEPA$_k$ extends PEPA with formal parameters, guards and value passing. It might at first seem that we can use parameter passing to describe the firing of a PEPA net and the communication of a token from one place to another. However, the parameters of a PEPA$_k$ component must have integer values. That is, the component definitions cannot describe *higher-order components*, which are themselves parameterised by another component. In PEPA$_k$, guards are used to restrict the evaluation of process expressions to those cases where the guard evaluates to *true* under the current assignment of (integer) values to parameters. In PEPA nets, cells are storage locations for components where the behaviour of the component is either fully enabled (because the cell contains a token) or fully disabled (because the cell is empty). Accordingly, we can use guarded components to model cells if we can write guards which evaluate to *true* exactly when the token is present and *false* otherwise.

Each PEPA net has a finite number of places and each token has a finite number of colours (local states). Because of this, we can encode the information that a token of a particular colour is at a particular place by a pair of integer variables. Call these variables `place` and `colour` respectively. We can now encode a PEPA net cell in place j as a sum of PEPA$_k$ guarded components as shown below.

$$\underbrace{P[\cdot]}_{\substack{\text{PEPA net cell at place } j,\\ \text{tokens } = \{P_0,\ldots,P_n\}}} \quad\leadsto\quad \underbrace{\sum_{i=0}^{n}[(\texttt{place} = j) \wedge (\texttt{colour} = i)] \Rightarrow P_i}_{\text{PEPA}_k \text{ expression in atomic submodel } j}$$

The Möbius implementation of a PEPA net is built up as a composition of atomic submodels, each implemented in PEPA$_k$, one for each place in the PEPA net.

To model the firing of a PEPA net where the place of a token and its colour are simultaneously updated we update the variables to store the new place and the new colour, e.g. (`place` = k, `colour` = c). The definition of the variables is placed at a higher level of the Möbius Replicate/Join tree to allow the variables to be shared between the atomic submodels.

To implement dynamically varying communication structure in PEPA$_k$ it would be necessary to replicate some components or rename some activities.

7 Case study: a peer-to-peer file system

We have used the PEPA Workbench for PEPA nets to analyse performance aspects of a modern networked application, Freenet [22], modelled as a PEPA net. The Freenet project began in the Division of Informatics of The University of Edinburgh. Its purpose is to develop a de-centralised networked application which allows users to publish information for retrieval over the network. Publications are replicated at a subset of the nodes in the network and requests to retrieve them cause them to be further replicated at other nodes nearer to the requestor. Little-accessed publications can be dropped from nodes. In this way, in time, a little-used publication could eventually disappear from the network entirely (it would "go out of print").

Up to this point one could simply think of Freenet as a probabilistic filestore but one striking novelty of the Freenet platform is that it attempts to ensure the anonymity of both authors and readers and the ability of node maintainers to deny knowledge of the contents stored at their node. Additional layers of indirection mean that it is not feasible to determine the origin of a file passing through the network. No centralised server is used to maintain a registry of publications on the network (this is a peer-to-peer architecture) so there is no central control to be held accountable for the contents of the network as a whole.

Clients built on Freenet include *Frost*. Frost is a file-sharing tool similar to Napster and Morpheus. It additionally provides a message board system that allows anonymous Usenet-style communication. Applications of this include enabling free speech in countries without freedom of speech laws.

Because the system architecture contains no centralised index, retrieving a file in Freenet consists of making a request to a Freenet host. This becomes a series of requests as one host interrogates another to satisfy the request. Files on Freenet are stored under descriptions which are hashed to provide search keys and searching for a file proceeds by steepest-ascent hill-climbing. To prevent infinite chains of requests, each request has a "hops-to-live" count which is decreased each time that it hops from one Freenet node to another. When the hops-to-live count reaches zero the search is abandoned and it is reported that the file could not be found.

The behaviour of a Freenet node is modelled as a static PEPA net component. The node first receives a request which comes from either the user or from another node. It checks its own store to see if the data is found. It returns the data if found, allowing another request to be initiated. If not found a further request, found/notfound iteration is initiated.

$$\begin{aligned}
\mathit{Node} &\overset{\mathit{def}}{=} (\mathit{request}, r).\mathit{Node}_2 \\
\mathit{Node}_2 &\overset{\mathit{def}}{=} (\mathit{found}, p \times f).\mathit{Node}_3 + (\mathit{notfound}, (1-p) \times f).\mathit{Node} \\
\mathit{Node}_3 &\overset{\mathit{def}}{=} (\mathit{return}, rt).\mathit{Node}
\end{aligned}$$

The request is a PEPA net token, moving around the network via the ***hop*** activity. We model failure due to exceeding the hops-to-live count (*die*). The subscript on a request counts down the number of hops left to live. The request dies if it has not found the file and has no more hops left. When it dies, the component becomes “recycled” as a new request (for a different file).

$$
\begin{array}{lll}
\mathit{Request}_h \stackrel{\mathit{def}}{=} (\mathit{request}, \top).\mathit{Request}'_h & & (h_{max} \geq h \geq 0) \\
\mathit{Request}'_h \stackrel{\mathit{def}}{=} (\mathit{found}, \top).\mathit{Request}'_{h_{max}} & & \\
\qquad + (\mathit{notfound}, \top).\mathit{Request}''_h & & (h_{max} \geq h \geq 0) \\
\mathit{Request}''_h \stackrel{\mathit{def}}{=} (\boldsymbol{hop}, \kappa).\mathit{Request}_{h-1} & & (h_{max} \geq h > 0) \\
\mathit{Request}''_0 \stackrel{\mathit{def}}{=} (\mathit{die}, d).\mathit{Request}_{h_{max}} & &
\end{array}
$$

Each place in the net combines a slot for a request with a node as shown below.

$$P_i[R] \stackrel{\mathit{def}}{=} \mathit{Request}[R] \underset{\{\,\mathit{request},\, \mathit{found},\, \mathit{notfound}\,\}}{\bowtie} \mathit{Node}$$

We have processed a modest-sized model of this system with the PEPA Workbench for PEPA nets and solved it with the Maple computer algebra system to find performance measures of the system. One such performance measure was the dependency of the number of requests on the probability of finding the file at a given node, subject to two simplifying assumptions of uniform request origins and uniform file distribution. We found the PEPA nets formalism to give a relatively natural means of expressing the structure of a system such as this one, where mobile components have local state which evolves at places in the net.

8 Further work

We have defined a language which provides an extension to the PEPA stochastic process algebra by allowing a number of distinct PEPA models to be arranged into a net. These models communicate via the transfer of tokens from one place to another. We have implemented this new language and applied it to some case studies. In the light of additional experience gained from further case studies it could be possible that we would discover that other language constructs would be helpful to the modeller.

One possibility would be an independent evolution of the net system, akin to the *transport transitions* of elementary object systems, where tokens are forcibly moved from one place to another without the option to refuse this or change state in transit. We have omitted this feature at present because it seems at odds with the process algebra notion of every component having behaviour. Additional language design decisions and extensions remain as future work.

Other future work includes the continued development of our implementation of the PEPA Workbench for PEPA nets. The additional of a graphical editor for PEPA nets is a likely next step with this tool.

Together with Norman and Parker at Birmingham we have recently extended the PRISM probabilistic symbolic model checker [23] to support the PEPA stochastic process algebra as an additional modelling language. An extension of that work to support PEPA nets would greatly enhance our ability to experiment with models on a large scale.

9 Conclusions

The PEPA nets formalism is new and, as yet, relatively unproven. It is our belief that it can provide a suitable framework for the description of performance models of systems which have distinct notions of changes of state. Our experience with the PEPA formalism has been that the combination of a well-defined formal semantics for the language and the availability of a range of tools to implement the language has enabled us and others to use it effectively in the performance modelling and analysis of systems. By following a similar development path we would hope that the PEPA nets formalism could also prove to be useful.

The combination of a process algebra with a Petri net presents many opportunities to import developments from the Petri net community into the practices in the process algebra community. Further, it is to be hoped that these developments can be imported more directly through the use of a Petri net with algebraic terms as tokens than if one was to rework them and to re-apply them in the process algebra context.

Acknowledgements

Stephen Gilmore and Jane Hillston are supported by the DEGAS (Design Environments for Global ApplicationS) project funded by the FET Proactive Initiative on Global Computing. Our thanks go to Salem Derisavi of the Möbius project for providing insights into the operation of PEPA$_k$ in Möbius. Thanks also to Giuliana Franceschinis for her advice on coloured Petri nets. The authors are grateful to the anonymous referees of the Tools conference for their comments on the paper. An earlier version of this paper appeared as [24].

References

1. M. Ajmone Marsan, A. Bobbio, and S. Donatelli. Petri nets in performance analysis: An introduction. In Reisig, W. and Rozenberg, G., editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 211–256. Springer-Verlag, 1998.
2. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
3. M. Bernardo and R. Gorrieri. A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.
4. H. Hermanns. *Interactive Markov Chains.* PhD thesis, Universität Erlangen-Nürnberg, 1999.
5. S. Donatelli, J. Hillston, and M. Ribaudo. A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham, North Carolina, 1995.
6. S. Gilmore, J. Hillston, and L. Recalde. Elementary structural analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh, 1997.

7. I. C. Rojas M. *Compositional construction and analysis of Petri net systems.* PhD thesis, The University of Edinburgh, 1997.
8. J. Hillston, L. Recalde, M. Ribaudo, and M. Silva. A comparison of the expressiveness of SPA and bounded SPN models. In B. Haverkort and R. German, editors, *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, Aachen, Germany, September 2001. IEEE Computer Science Press.
9. C. Reinke. Haskell-Coloured Petri Nets. In *Implementation of Functional Languages, 11th International Workshop*, volume 1868 of *LNCS*, pages 165–180, Lochem, The Netherlands, September 1999. Springer-Verlag.
10. M. Becker and H. Szczerbicka. PNiQ: Integration of queuing networks in generalized stochastic petri nets. *IEE Proceedings—Software*, 146(1):27–33, February 1999. Special issue of the proceedings of the Fourteenth UK Performance Engineering Workshop.
11. F. Bause. Queueing Petri nets—a formalism for the combined qualitative and quantitative analysis of systems. In *5th International Workshop on Petri Nets and Performance Models*, pages 14–23, Toulouse, France, October 1993.
12. B.R. Haverkort, I.G. Niemegeers, and P Veldhuyzen van Zanten. DyQNtool—a performability modelling tool based on the dynamic queueing network concept. In G. Balbo and G. Serazzi, editors, *Modelling Techniques and Tools for Computer Performance Evaluation*, pages 181–195. North-Holland, 1992.
13. R. Valk. Petri nets as token objects—an introduction to Elementary Object Nets. In J. Desel and M. Silva, editors, *Proceedings of the 19th International Conference on Application and Theory of Petri Nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25, Lisbon, Portugal, 1998. Springer-Verlag.
14. C. Priami. Stochastic π-calculus. In S. Gilmore and J. Hillston, editors, *Proceedings of the Third International Workshop on Process Algebras and Performance Modelling*, pages 578–589. Special Issue of *The Computer Journal*, 38(7), December 1995.
15. K.G. Larsen. Compositional theories based on an operational semantics of contexts. In *REX Workshop on Stepwise Refinement of Parallel Systems*, volume 430 of *LNCS*, pages 487–518. Springer-Verlag, May 1989.
16. G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models.* PhD thesis, The University of Edinburgh, 2000.
17. J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.
18. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.
19. G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001.
20. R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML: Revised 1996.* The MIT Press, 1996.
21. G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 200–215, Aachen, Germany, September 2001. Springer-Verlag.

22. I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66, Berkeley, California, 2001. Springer-Verlag.
23. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. This volume, 2002.
24. S. Gilmore, J. Hillston, and M. Ribaudo. PEPA-coloured stochastic Petri nets. In K. Djemame and M. Kara, editors, *Proceedings of the Seventeenth UK Performance Engineering Workshop*, pages 155–166, University of Leeds, July 2001.

A Summary of the PEPA language

The PEPA language provides a small set of combinators. These allow language terms to be constructed defining the behaviour of components, via the activities they undertake and the interactions between them. The syntax may be formally introduced by means of the grammar which was shown in Figure 1. In that grammar 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 or a model component, as defined by a defining equation. C when subscripted with an S stands for constants which denote sequential components. The component combinators, together with their names and interpretations, are presented informally below.

Prefix: The basic mechanism for describing the behaviour of a system is to give a component a designated first action using the prefix combinator, denoted by a full stop. For example, the component $(\alpha, r).S$ carries out activity (α, r), which has action type α and an exponentially distributed duration with parameter r, and it subsequently behaves as S. Sequences of actions can be combined to build up a life cycle for a component.

Choice: The life cycle of a sequential component may be more complex than any behaviour which can be expressed using the prefix combinator alone. The choice combinator captures the possibility of competition between different possible activities. 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.

Hiding: The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted by the division sign in P/L. Here, the set L of visible action types identifies those activities which are to be considered internal or private to the component. These activities are not visible to an external observer, nor are they accessible to other components for cooperation. Once an activity is hidden it only appears as the unknown type τ; the rate of the activity, however, remains unaffected.

Cooperation: Most systems are comprised of several components which interact. In PEPA direct interaction, or *cooperation*, between components is represented by the butterfly combinator. The set which is used as the subscript to the cooperation symbol determines those activities on which the *cooperands* are forced to synchronise. Thus the cooperation combinator is in fact an indexed family of combinators, one for each possible *cooperation set* L (we write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty). When cooperation is not imposed, namely for action types not in L, the components proceed independently and concurrently with their enabled activities. However 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.* The rate of the shared activity may be altered to reflect the work carried out by both components to complete the activity.

In some cases, when an activity is known to be carried out in cooperation with another component, a component may be *passive* with respect to that activity. This means that the rate of the activity is left unspecified and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.

Model components capture the structure of the system in terms of its *static* components. The dynamic behaviour of the system is represented by the evolution of these components, either individually or in cooperation. The form of this evolution is governed by a set of formal rules which give an operational semantics of PEPA terms. The semantic rules, in the structured operational style, are presented in Figure 7 without further comment; the interested reader is referred to [2] for more details. The rules are read as follows: if the transition(s) above the inference line can be inferred, then we can infer the transition below the line. The notation $r_\alpha(E)$ which is used in the third cooperation rule denotes the apparent rate of α in E.

Thus, as in classical process algebra, the semantics of each term in PEPA is given via a labelled *multi-transition* system—the multiplicities of arcs are significant. In the transition system a state corresponds to each syntactic term of the language, or *derivative*, and an arc represents the activity which causes one derivative to evolve into another. The complete set of reachable states is termed the *derivative set* of a model and these form the nodes of the *derivation graph* formed by applying the semantic rules exhaustively.

The timing aspects of components' behaviour are not represented in the states of the derivation graph, but on each arc as the parameter of the negative exponential distribution governing the duration of the corresponding activity. The interpretation is as follows: when enabled an activity $a = (\alpha, r)$ will delay for a period sampled from the negative exponential distribution which has pa-

Prefix

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

Cooperation

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

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L) \qquad \text{where } R = \frac{r_1}{r_\alpha^E}\frac{r_2}{r_\alpha^F}\min(r_\alpha(E), r_\alpha(F))$$

Choice

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

Hiding

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

Constant

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

Fig. 7. The operational semantics of PEPA

rameter r. If several activities are enabled concurrently, either in competition or independently, we assume that a *race condition* exists between them. Thus the activity whose delay before completion is the least will be the one to succeed. The evolution of the model will determine whether the other activities have been *aborted* or simply *interrupted* by the state change. In either case the memoryless property of the negative exponential distribution eliminates the need to record the previous execution time.

When two components carry out an activity in cooperation the rate of the shared activity will reflect the working capacity of the slower component. We assume that each component has a capacity for performing an activity type α, which cannot be enhanced by working in cooperation (it still must carry out its own work), unless the component is passive with respect to that activity type. For a component P and an action type α, this capacity is termed the *apparent rate* of α in P. It is the sum of the rates of the α type activities enabled in P. The apparent rate of α in a cooperation between P and Q over α will be the minimum of the apparent rate of α in P and the apparent rate of α in Q.

The derivation graph is the basis of the underlying Continuous Time Markov Chain (CTMC) which is used to derive performance measures from a PEPA model. The graph is systematically reduced to a form where it can be treated as the state transition diagram of the underlying CTMC. Each derivative is then a state in the CTMC. The *transition rate* between two derivatives P and Q in the derivation graph is the rate at which the system changes from behaving as component P to behaving as Q. It is denoted by $q(P, Q)$ and is the sum of the activity rates labelling arcs connecting node P to node Q. In order for the CTMC to be *ergodic* its derivation graph must be strongly connected. Some necessary conditions for ergodicity, at the syntactic level of a PEPA model, have been defined [2]. These syntactic conditions are imposed by the grammar introduced earlier.

A.1 Definition of PEPA nets equality on alphabets

The relation $=_a$ is used in the PEPA nets semantics. Its definition is straightforward but is included here for completeness.

$$P =_a Q \quad \text{if} \quad \operatorname{alph} P = \operatorname{alph} Q$$

The alphabet of a PEPA nets component is the least set satisfying the following equations.

$$\begin{aligned}
\operatorname{alph}(P \underset{L}{\bowtie} Q) &= ((\operatorname{alph} P) \setminus L) \cup ((\operatorname{alph} Q) \setminus L) \cup ((\operatorname{alph} P) \cap L \cap (\operatorname{alph} Q)) \\
\operatorname{alph}(P/L) &= (\operatorname{alph} P) \setminus L \\
\operatorname{alph}(P[C]) &= \operatorname{alph} P \\
\operatorname{alph} I &= \operatorname{alph} S \text{ where } I \overset{\mathit{def}}{=} S \\
\operatorname{alph}((\alpha, r).S) &= \{\, \alpha \,\} \cup \operatorname{alph} S \\
\operatorname{alph}(R + S) &= \operatorname{alph} R \cup \operatorname{alph} S
\end{aligned}$$

A.2 Availability of the modelling tools

The PEPA modelling tools, together with user documentation and papers and example PEPA models are available from the PEPA Web page which is located at `http://www.dcs.ed.ac.uk/pepa`. The PEPA Workbench for PEPA nets is available for download from this page in versions for Linux, Solaris and Windows.