

PEPA nets

Stephen Gilmore, Jane Hillston, and Leïla Kloul*

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

Abstract. In this chapter 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 linking 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 demonstrate the modelling capabilities of the formalism on a number of examples, including a mobile server running MobileIP.

1 Introduction

Over the last decade mobility has had a major impact on the way we design, implement and manage many computer systems. Mobility may be manifest in the form of devices which change location and spontaneously connect/disconnect, or in the form of executable code which is moved around the network for a variety of reasons. In either case the effect is that the context in which computation is taking place is dynamically changing, and these changes will have consequences for the performance of the system. In this chapter we introduce the modelling formalism PEPA nets which have been designed to capture information about mobility and so allow performance models of such systems be readily and naturally developed.

The rest of the chapter is organised as follows.

Section 2 introduces the notation and terminology of PEPA nets, after a brief introduction to PEPA. (Readers are assumed to be familiar with the basic ideas of Stochastic Petri Nets, and are referred to [1] for basic definitions.) In Section 3 we present two small examples, a simple mobile agent system, and a secure web service, all modelled as PEPA nets. Section 4 is a more detailed case study of a mobile host accessible via MobileIP. In Section 5 we discuss tool support for PEPA nets. Related work is discussed in Section 6. Concluding remarks and further work are presented in Section 7.

2 PEPA nets

In this section we present the concepts and definitions used in PEPA nets. First we give an intuitive overview of the formalism, followed by a more detailed account of its definition and its formal properties.

* L. Kloul is on leave from PRISM, Université de Versailles, 45, Av. des Etats-Unis 78000 Versailles, France.

A PEPA net is a stochastic Petri net with coloured tokens. The tokens represent *mobile* objects with state and behaviour, where we use the term *mobile* loosely to characterise objects which may find themselves in different contexts during execution. The tokens are described using a stochastic process algebra, Hillston’s Performance Evaluation Process Algebra (PEPA).

The use of stochastic Petri nets for performance models is well-established [1] and coloured variants, e.g. Stochastic Well-Formed Nets (SWN) [?], have also been developed. However the use of colours in PEPA nets offers something quite distinct — the possibility of differentiating between two types of change of state within a system. Unlike SWN where tokens remain indistinguishable within their colour classes, tokens within PEPA nets are autonomous components. Firings of the net will typically be used to model macro-step (or *global*) changes of state, whereas transitions within the PEPA tokens are typically used to model micro-step (or *local*) changes of state as components undertake activities. Thus modelling with PEPA nets uses both Petri nets and process algebras together as a single, structured performance modelling formalism. There is some reason to believe that these two formalisms complement each other [10]. In particular, we have previously demonstrated [13] that PEPA nets offer some expressivity which is not directly offered by either PEPA or Petri nets.

2.1 Summary of the PEPA language

In the following paragraphs we give a brief overview of PEPA. Readers are referred to [18] for a more detailed introduction.

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. In particular we develop a model as a *model component* constructed using *static combinators*, from a number of *sequential components* constructed using *dynamic combinators*.

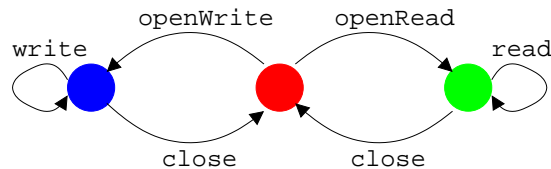


Fig. 1. Simple file protocol

Consider a `File` class with methods `openRead()`, `openWrite()`, `read()`, `write()` and `close()`. The order in which the methods can be applied defines a *protocol* for a `File` object. This could be represented as shown in Figure 1. We can express this as a PEPA component as below:

$$\begin{aligned}
 File &\stackrel{\text{def}}{=} (openRead, r_o).InStream + (openWrite, r_o).OutStream \\
 Instream &\stackrel{\text{def}}{=} (read, r_r).File + (close, r_c).File \\
 Outstream &\stackrel{\text{def}}{=} (write, r_w).File + (close, r_c).File
 \end{aligned}$$

Here $\stackrel{\text{def}}{=}$ denotes definitional equality and a *constant* or *identifier* (e.g. *File*) is used to assign a name to a pattern of behaviour associated with a component. Each activity such as $(openRead, r_o)$ has an *action type* (*openRead*) and an exponentially distributed duration, denoted by its parameter (r_o). The basic mechanism for describing the behaviour of a system is to give a component a designated first action using the *prefix* combinator, denoted “.”. However, the life cycle of a component may be more complex than any behaviour which can be expressed using the prefix combinator alone. The choice combinator, denoted by $+$, captures the possibility of competition between different possible activities: only one will succeed and the other alternative will be discarded (for the moment). The competition is resolved via the race policy.

File is a sequential component and prefix and choice are the dynamic combinators.

Now consider a `File` being accessed by a `FileReader` object. The two components must interact via the file protocol. In PEPA this is would be denoted as

$$File \underset{L}{\bowtie} FileReader$$

where L denotes the set of action types $\{openRead, read \text{ and } close\}$. The combinator $\underset{L}{\bowtie}$ denotes *cooperation* between the components. This is the basis of compositionality within PEPA. The set which is used as the subscript of the cooperation symbol, the *cooperation set* L , determines those activities on which the two components are forced to synchronise. 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 (for details see [18]). We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty.

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 (denoted \top) 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.

In the case of the `File` object cooperating with a `FileReader` we may wish to impose that the `File` object may only be read by this `FileReader`. This is denoted using the *hiding* combinator, $/$:

$$(File \underset{L}{\bowtie} FileReader) / \{read\}$$

Hiding provides the possibility to abstract away some aspects of a component’s behaviour. In a component P/L , the set L of visible action types will be considered internal or private to the component and will appear as the unknown type τ . Hiding and cooperation of the dynamic combinators.

The syntax of PEPA may be formally introduced by means of the grammar shown in the lower part of Figure 2. In that grammar S denotes a *sequential component* and P denotes a *model component* which executes in parallel. I stands for a constant which

denotes either a sequential or a model component, as defined by a defining equation. 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 8 in the Appendix.

The semantics of each term in PEPA is given via a labelled transition system. 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* which is formed by applying the semantic rules exhaustively.

The timing aspects of components' behaviour are represented 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 parameter r . If several activities are enabled concurrently, either in competition or independently, we assume that a *race condition* exists between them. The evolution of the model will determine whether the other activities have been *aborted* or simply *interrupted* by the resulting state change. In either case the memoryless property of the 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, 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* [18] 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 [18]. These syntactic conditions are imposed by the grammar in Figure 2.

2.2 Introduction to PEPA nets

In outlining a framework of design paradigms for mobile code systems in [11], Fuggetta *et al.* emphasise three architectural concepts: *components*, *interactions* and *sites*. The importance of including an explicit notion of *location* has also been recently been highlighted by Köhler *et al.* [21], who additionally choose *entities* and *movement* as prim-

itives. In PEPA nets the places of the PEPA net represent sites or locations, the PEPA components represent components or entities, interactions are represented by PEPA cooperations, which may only take place between co-located components, and movement is represented by firings of the PEPA net which move a token from one place to another. Thus in PEPA nets we have two distinct types of change of state: *firings* and *transitions*.

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 subsequent evaluation both in the source (where existing cooperations with other components now can no longer take place) and in the target (where previously disabled cooperations 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 cooperation with another component). Components can only cooperate if they are resident in the same place in the net. The PEPA net formalism does not allow components at different places in the net to cooperate 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. Additionally we do not allow a firing to coincide with a transition which is shared, i.e. it is not possible for two components in one place to cooperate *and* transfer to another place as an atomic action. Thus transitions in a PEPA net have local effect because they involve only components at one place in the net.

There are distinct alphabets for transitions and firings, meaning that the same action type cannot be used for both. Thus there can be no ambiguity between them.

A PEPA net is made up of PEPA *contexts*, one at each place in the net. A context consists of a number of *static* components (possibly zero) and a number of *cells* (at least one). 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. For example, if we consider the *FileReader* introduced earlier, it might reside in a fixed location of the system and interact with any files copied into its scope. This would be denoted

$$File[-] \underset{t}{\bowtie} FileReader$$

The components which fill cells can circulate as the tokens of the net. In contrast, the static components cannot move. Most variants of Petri nets do not include static tokens, the closest concept being “self loops” where a token is deleted from a place and then immediately replaced. Here static components provide the infrastructure of the place and act as cooperation partners in synchronisation activities with tokens. Contexts have previously been used in both classical process algebras [24], and in the stochastic process algebra PEPA [8].

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

the same transitions as Q . $Q[Q']$ enables the same transitions as Q' , etc. As usual with PEPA components we require that the component has an ergodic definition so that it is always possible to return to a state which one has previously reached.

We use capitalised names to denote PEPA components (such as P and Q) and lowercase for PEPA transitions (such as a and b). We use bold capitalised names for PEPA net places (such as \mathbf{P}_1 and \mathbf{P}_2) and bold lowercase for PEPA net firings (such as \mathbf{a} and \mathbf{b}).

Note that the expression of the structural information contained in a PEPA net could be represented by any transition-based modelling formalism. Indeed it would be possible to use a PEPA component to control the possible “firings” (macro-steps) of the model. However, we feel that there are some advantages to using a Petri net in this role. Firstly, using a different formalism gives a clearer separation of concerns within our model making it both easier to construct and to understand. Furthermore, this macro-level is often of a size that can benefit from graphical representation, to give an intuitive understanding of the coarse structure of the model. Finally, the *movement* of components—to a new host, to a new context, etc.—has resonance with the systems we study.

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 to record their number and one could present the marking of a Petri net with places P_1 , P_2 and P_3 as $(P_1 : 2, P_2 : 1, P_3 : 0)$. 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 \mathbf{P}_1 , \mathbf{P}_2 and \mathbf{P}_3 as shown below.

$$\begin{aligned}\mathbf{P}_1[Q] &\stackrel{\text{def}}{=} Q[Q] \underset{L}{\boxtimes} R \\ \mathbf{P}_2[Q] &\stackrel{\text{def}}{=} Q[Q] \underset{K}{\boxtimes} S \\ \mathbf{P}_3[Q] &\stackrel{\text{def}}{=} Q[Q] \underset{K \cup L}{\boxtimes} (R \parallel S)\end{aligned}$$

From its use in the contexts at each place we see that Q is a component which can move as a token around the net whereas R and S are static components which cannot move. There is a copy of R at place \mathbf{P}_1 and another at \mathbf{P}_3 . There is a copy of S at place \mathbf{P}_2 and another at \mathbf{P}_3 . Clearly the marking of a PEPA net needs to record the current location of the tokens circulating in the net, and their current state. However, this is not sufficient to establish the state of the system — the local state captured by each place’s marking will also depend on the current state of the in the place. Thus to identify these states we allow place definitions to specify a particular state of each of the static components. Thus, in the example above, if S can evolve to S' we can define $\mathbf{P}'_2[Q] \stackrel{\text{def}}{=} Q[Q] \underset{K}{\boxtimes} S'$.

Net-level transitions in a PEPA net A labelling function ℓ is used to associate an activity, consisting of an action type and a rate, (α, r) , with each net level transition t .

Note that 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 conjunction with the component which is performing it. For a firing to be enabled there must be a token in the input place of the net-level transition, that token must enable the corresponding activity, and there must be an empty cell at the output place of the transition of the correct token type.

As an example, suppose that Q is a component which is currently at place \mathbf{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 \mathbf{P}_1 and \mathbf{P}_2 labelled by (α, r_2) . If Q performs activity α in this setting it will be removed from \mathbf{P}_1 (leaving behind an empty cell) and Q' will be deposited into \mathbf{P}_2 (filling an empty cell there).

A priority function π maps action types to the natural numbers, and can be used to eliminate some firings from the labelled multi-transition system: only enabled firings with the highest priority value are considered eligible to fire. For example, suppose that Q is a component which is currently at place \mathbf{P}_1 and that it can perform activities of types α, β and γ where $\pi(\alpha) = \pi(\beta) = 2$ whereas $\pi(\gamma) = 1$. Further, suppose that there are net transitions between \mathbf{P}_1 and each of $\mathbf{P}_2, \mathbf{P}_3$ and \mathbf{P}_4 labelled by α, β and γ respectively. Assuming that there are appropriate empty cells in all places, Q may perform activity α and be deposited in place \mathbf{P}_2 , or activity β and be deposited in place \mathbf{P}_3 but it cannot perform activity γ and be deposited in place \mathbf{P}_4 . Only if there are no empty cells in places \mathbf{P}_2 and \mathbf{P}_3 will activity γ become enabled.

Net structure of a PEPA net The class of nets that we currently use for modelling the net structure of a PEPA net is restricted to *structural state machines*, i.e. nets whose transitions can have only one input place and one output place¹. This means that we can represent conflicts at the net level, while synchronisations are not allowed. This is consistent with the fact that PEPA components cannot cooperate on a shared activity when they are resident in different places.

It is usual with coloured Petri nets to associate functions with arcs, offering a generalisation of the usual, basic “functions” offered by arc multiplicities. In PEPA nets the arc functions are implicit. The modification of a token which takes place when it is fired is wholly specified by the action type of the firing, the definition of the token and the semantics. Furthermore, although we allow multiple tokens within net places, only one token can move at each firing. Thus arc multiplicities greater than one are not allowed.

Formal definition The introduction of contexts requires an extension to the syntax of PEPA. This extension is presented in Figure 2.

We assume that there is a set \mathcal{A} of PEPA action types which can be partitioned into disjoint subsets \mathcal{A}_f and \mathcal{A}_t corresponding to firings and local transitions respectively.

Definition 1. A PEPA net \mathcal{N} is a tuple $\mathcal{N} = (\mathcal{P}, \mathcal{T}, I, O, \ell, \pi, \mathcal{C}, D, M_0)$ such that

¹ This restriction has recently been relaxed by allowing more general net structures [14].

$$\begin{array}{l}
N ::= D^+ M \quad (\text{net}) \\
\text{(definitions and marking)} \\
\\
M ::= (M_{\mathbf{P}}, \dots) \quad (\text{marking}) \quad D ::= I \stackrel{\text{def}}{=} S \quad (\text{component defn}) \\
M_{\mathbf{P}} ::= \mathbf{P}[C, \dots] \quad (\text{place marking}) \quad | \quad \mathbf{P}[C] \stackrel{\text{def}}{=} P[C] \quad (\text{place defn}) \\
\text{(marking vectors)} \quad | \quad \mathbf{P}[C, \dots] \stackrel{\text{def}}{=} P[C] \boxtimes_L P \quad (\text{place defn}) \\
\text{(identifier declarations)} \\
\\
S ::= (\alpha, r).S \quad (\text{prefix}) \quad P ::= P \boxtimes_L P \quad (\text{cooperation}) \quad C ::= ' ' \quad (\text{empty}) \\
| S + S \quad (\text{choice}) \quad | P/L \quad (\text{hiding}) \quad | S \quad (\text{full}) \\
| I \quad (\text{identifier}) \quad | P[C] \quad (\text{cell}) \\
\text{(sequential components)} \quad | I \quad (\text{identifier}) \\
\text{(concurrent components)} \quad \text{(cell term expressions)}
\end{array}$$

Fig. 2. The syntax of PEPA extended with contexts

- \mathcal{P} is a finite set of places;
- \mathcal{T} is a finite set of net transitions;
- $I : \mathcal{T} \rightarrow \mathcal{P}$ is the input function;
- $O : \mathcal{T} \rightarrow \mathcal{P}$ is the output function;
- $\ell : \mathcal{T} \rightarrow (\mathcal{A}_f, \mathbb{R}^+ \cup \{\top\})$ is the labelling function, which assigns a PEPA activity ((type, rate) pair) to each transition. The rate determines the negative exponential distribution governing the delay associated with the transition;
- $\pi : \mathcal{A}_f \rightarrow \mathbb{N}$ is the priority function which assigns priorities (represented by natural numbers) to firing action types;
- $\mathcal{C} : \mathcal{P} \rightarrow P$ is the place definition function which assigns a PEPA context, containing at least one cell, to each place;
- D is the set of token component definitions;
- M_0 is the initial marking of the net.

PEPA nets are governed by the semantic rules for PEPA and few additional rules — both sets are provided in Appendix ???. Informally the new rules can be interpreted as follows:

- The Cell rule defines that a cell which is filled by a component Q is able to make the same transitions as Q itself. 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 and that these transitions are exactly the transitions generated by the PEPA semantics (including the extension for contexts).

- 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. In order for a firing to take place it must be the case that the type of the enabled firing has the highest priority level in the set of enabled firings. In other words, for a firing to occur there must not be any other firing satisfying the Enabling rule (empty destination cell) which has a higher priority.
- The Enabling rule determines when a transition is considered to be enabled. In addition to ensuring that there is a valid token to fire we must also check that there is an empty cell in the destination place into which the token can be transferred. The Enabling rule ensures that this is the case, and defines a transition relation, decorated with the priority level of the corresponding activity type. The rate at which the activity is enabled is calculated as in the PEPA semantics of cooperation.

From the operational semantics a derivation graph and underlying CTMC can be extracted from any PEPA net, and this is the basis on which performance analysis is carried out.

The net bisimulation relation PEPA nets are equipped with an equivalence relation called *net bisimulation*. This relation is important both in theory and in practice. In the evolution of the state space of a model by our tool we only store states up to net bisimulation, i.e. we carry out automatic aggregation over equivalent states. This provides a dramatic reduction in the state space of the model under certain conditions.

Our relation is defined in the style of Larsen and Skou [23], based on a conditional transition rate between markings. The *conditional transition rate* from marking M to marking M' via action type α , denoted $q(M, M', \alpha)$, is the sum of the activity rates labelling arcs connecting the corresponding nodes in the derivation graph which are labelled by the action type α . The *total conditional transition rate* from a marking M to a set of markings E is defined as

$$q[M, E, \alpha] = \sum_{M' \in E} q(M, M', \alpha)$$

Definition 2. An equivalence relation over markings, $\mathcal{R} \subseteq M \times M$, is a net bisimulation if whenever $(M, M') \in \mathcal{R}$ then for all $\alpha \in \mathcal{A}$ and for all equivalence classes $E \in M/\mathcal{R}$,

$$q[M, E, \alpha] = q[M', E, \alpha]$$

3 Examples

3.1 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 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. In this example the activities which can cause a firing of the net are **go** and **return**.

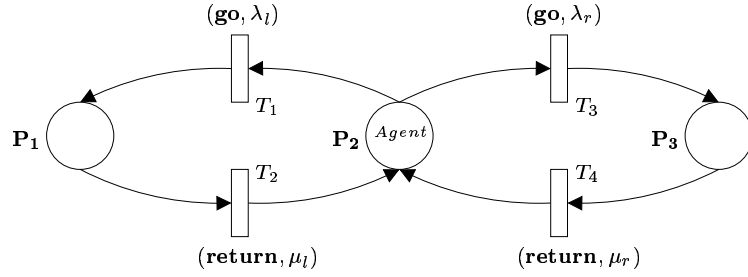


Fig. 3. A simple mobile agent system

Formally, we define the places of the net as shown in the PEPA context definitions below. We denote the local state of the context \mathbf{P}_2 by \mathbf{P}'_2 . This local state is arrived at when the static component *Master* has evolved to *Master'*.

$$\begin{aligned} \mathbf{P}_1[Agent] &\stackrel{\text{def}}{=} Agent[Agent] \underset{\{\text{interrogate}\}}{\boxtimes} Probe \\ \mathbf{P}_2[Agent] &\stackrel{\text{def}}{=} Agent[Agent] \underset{\{\text{dump}\}}{\boxtimes} Master \\ \mathbf{P}'_2[Agent] &\stackrel{\text{def}}{=} Agent[Agent] \underset{\{\text{dump}\}}{\boxtimes} Master' \\ \mathbf{P}_3[Agent] &\stackrel{\text{def}}{=} Agent[Agent] \underset{\{\text{interrogate}\}}{\boxtimes} Probe \end{aligned}$$

The initial marking of the net is $(\mathbf{P}_1[-], \mathbf{P}_2[Agent], \mathbf{P}_3[-])$ The behaviour of the components is given by the following PEPA definitions.

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

The derivation of the transition system underlying the model (Figure 4) gives us its underlying CTMC. This CTMC is solved for its stationary distribution and performance measures are calculated from that.

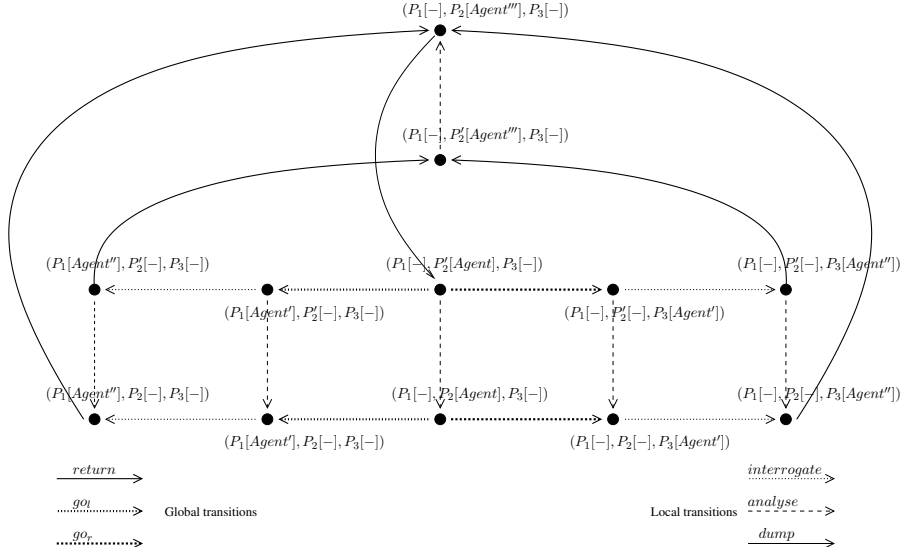


Fig. 4. The transition system of the mobile agent example

3.2 Secure Web Service

Our second example is a model of a mobile object system where a client sends SOAP message objects to a remote Web service. Our scenario is that a financial tycoon is sending requests for stocks and share price information to a remote Web service which provides this information. These requests for information are encrypted. An eavesdropper could make use of the information in the messages if they were sent as clear text. The Web service itself is protected by a firewall.

The token type The tokens exchanged in the system are SOAP messages in various formats. These may either be sent across the network as clear text or encrypted to preserve their contents. A SOAP message may be parsed to build an in-memory data structure which can be read and modified as needed. This data structure is a DOM tree (a Document Object Model tree).

$$\begin{aligned}
 \text{SoapMessage} &\stackrel{\text{def}}{=} (\text{send}_{cTr}, r_{sc}).\text{SentClearMessage} \\
 &\quad + (\text{encrypt}, r_e).\text{EncryptedMsg} \\
 &\quad + (\text{parse}, r_p).\text{DOMtree} \\
 \text{SentClearMessage} &\stackrel{\text{def}}{=} (\mathbf{\text{copyClear}}, \top).\text{SoapMessage}
 \end{aligned}$$

Encrypted messages can be decrypted to recover their initial contents or sent across the network in encrypted form.

$$\text{EncryptedMsg} \stackrel{\text{def}}{=} (\text{decrypt}, r_d).\text{SoapMessage}$$

$$\begin{aligned}
& + (sendenc, r_{se}).SentEncMessage \\
SentEncMessage & \stackrel{def}{=} (\mathbf{copyEncrypted}, \top).EncryptedMsg
\end{aligned}$$

In both cases, we model the transmission of a SOAP message as a two-phase process, separating the cost of making the decision to send ($send_{clr}$, $sendenc$) from the cost of copying the bytes across the network (**copyClear**, **copyEncrypted**).

DOM trees may be read or modified. As an in-memory data structure they first must be serialised (using the *export* activity) if they are to be sent across the network.

$$\begin{aligned}
DOMtree & \stackrel{def}{=} (read, r_r).DOMtree \\
& + (modify, r_m).DOMtree \\
& + (export, r_x).SoapMessage
\end{aligned}$$

Static components SOAP message tokens of various forms are exchanged between the places of the net. Static components at these places interact with the tokens. On the client side is the user, making requests of the remote Web Service. The user encrypts requests before they are sent and decrypts replies when they are received.

$$\begin{aligned}
User & \stackrel{def}{=} Encrypt + Decrypt \\
Encrypt & \stackrel{def}{=} (encrypt, \top).(sendenc, \top).User \\
Decrypt & \stackrel{def}{=} (decrypt, \top).(parse, \top).(read, \top).Request \\
Request & \stackrel{def}{=} (modify, \top).(export, \top).User
\end{aligned}$$

Running on the firewall is a gatekeeper process which performs three distinct functions: decrypting user requests, bouncing flawed requests and encrypting replies from the server. The gatekeeper receives requests from the user and decrypts them. The decrypted message might be a well-formed request, in which case it is forwarded to the server. Alternatively it might be in an invalid format, request a non-existent service, or have suspicious attachments. In this case it is bounced back to the user with a diagnostic error message attached. This decomposition of responsibilities means that the load on the server is reduced.

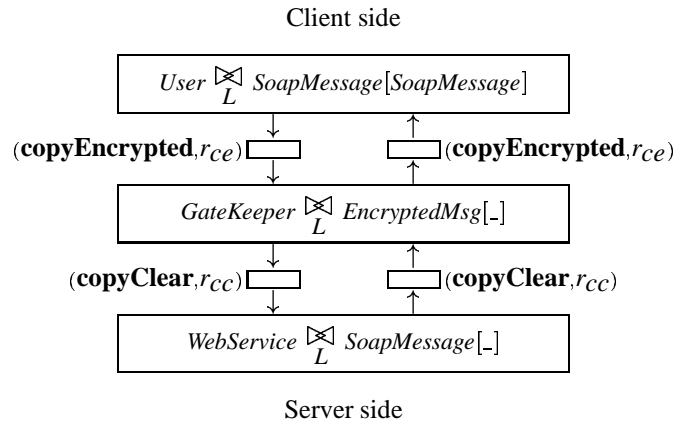
All of the communication with the user is sent in encrypted format. Behind the firewall the messages are exchanged in the clear. Thus $sendenc$ always sends to the user and $send_{clr}$ always sends to the server.

$$\begin{aligned}
GateKeeper & \stackrel{def}{=} FilterIn + Bounce + FilterOut \\
FilterIn & \stackrel{def}{=} (decrypt, \top).(send_{clr}, \top).GateKeeper \\
Bounce & \stackrel{def}{=} (decrypt, \top).FilterOut \\
FilterOut & \stackrel{def}{=} (encrypt, \top).(sendenc, \top).GateKeeper
\end{aligned}$$

Behind the firewall the share price Web service runs on the server. Its life cycle is parsing, reading, modifying, serialising and returning requests.

$$\begin{aligned}
WebService & \stackrel{def}{=} (parse, \top).(read, \top). \\
& (modify, \top).(export, \top).(send_{clr}, \top).WebService
\end{aligned}$$

The PEPA net The PEPA net of the system sites the above static components at places of the net and specifies the communication between different places of the net by naming the transitions which must be fired for tokens to move from place to place². The *User* is operating the client machine, the *GateKeeper* process runs on the firewall, the *WebService* on the server behind the firewall.



The synchronisation set used at each place, L , is $\{ \text{decrypt}, \text{send}_{cl}, \text{parse}, \text{read}, \text{modify}, \text{export}, \text{encrypt}, \text{send}_{enc} \}$.

4 Case study: Mobile IP

In this section we present a larger case study of modelling with PEPA nets. Based on the Internet protocol, Mobile IP is a standard protocol that makes user mobility transparent to applications and higher level protocols like TCP. It allows a *mobile node* to freely roam between network links and to remain always accessible. To achieve this the mobile node uses two IP addresses: a *home address* which is statically assigned on its *home network* and a *care-of address* which changes at each new point of attachment. On the home network, a proxy known as a *home agent* is responsible for forwarding all packets which are addressed to the mobile node on to its current care-of address.

Whenever the mobile node moves, it sends to its home agent a binding update message containing its home address, its current care-of address and the lifetime for which the binding should be honoured [20]. The home agent may refresh a binding cache entry by regularly requesting the transmission of the latest care-of address.

When the mobile node receives, via its home agent, a packet from a correspondent, it sends a binding update message to this correspondent. The correspondent may maintain a binding cache allowing its transmit function to redirect the packets to the mobile node's current care-of address [20]. The mobile node maintains a list of all its current correspondents and has to send them a binding update message each time it changes its point of attachment. Figure 5 summarises the main steps of the Mobile IP protocol.

² Note that for convenience we represent places as large rectangular boxes.

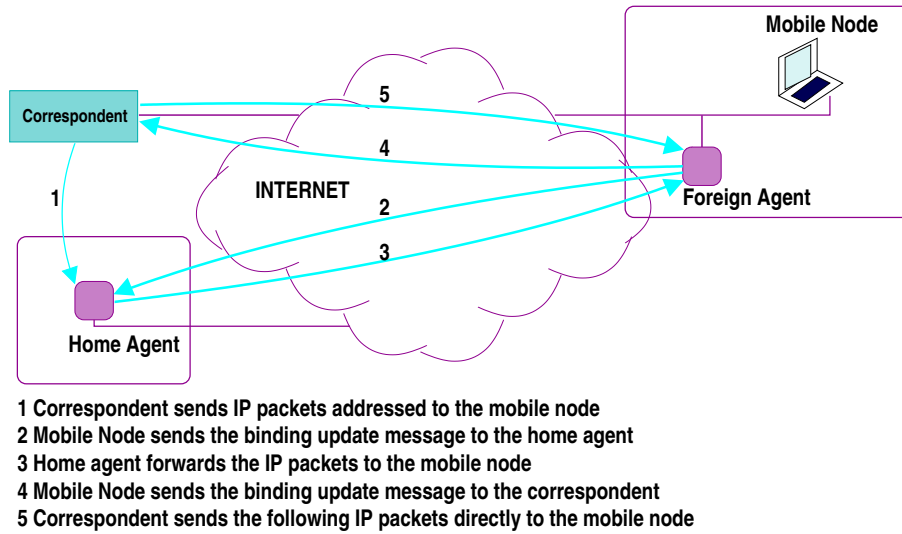


Fig. 5. The Mobile IP Protocol

In this study we assume that the system is composed of N domain or network hosts, besides the home network and the correspondent network. We assume that the home agent sends requests to the mobile node to update the care-of address. Once the correspondent has the care-of address, its transmit function redirects the packet to this address, saving one network hop relative to the route through the home agent [20].

4.1 The PEPA net Model

The system is modelled using the PEPA net model depicted in Figure 6 where the home network of the mobile node is represented using a place called **HOME_M**. The networks to which the mobile node may move are modelled by places **DOMAIN_{*i*}** where $i = 1, \dots, N$. Place **HOME_C** models the home network of the mobile node's correspondent.

Note that for the sake of readability in Figure 6 the rates of the activities labelling the firings are omitted. Moreover only the arcs between **DOMAIN₁** and **HOME_M** on one hand, and **HOME_C** on the other, are depicted. The arcs between the other domains and **HOME_M** and **HOME_C** are analogous.

To model the part of the protocol which manages the interaction between the home agent and the mobile node during its stay in place **DOMAIN_{*i*}**, we use two components *ProtoMA* and *CommuMA*. Similarly, components *ProtoMC* and *CommuMC* are used to model the protocol interactions between the mobile node and the correspondent, and *ProtoAC* and *CommuAC* for the exchanges between the home agent and the correspondent. Components *Mobile*, *Agent* and *Corresp* model the behaviour of the mobile node, the home agent and the current correspondent of the mobile node respec-

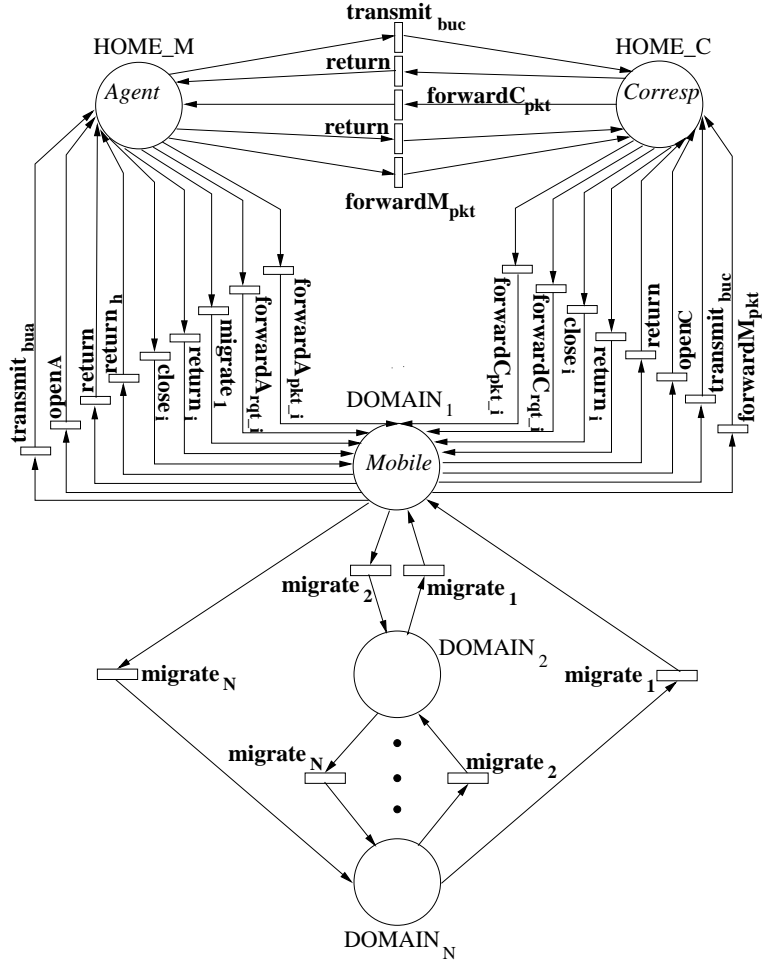


Fig. 6. The PEPA net model

tively. In contrast to *Mobile*, the last two components are static. All these components and places are explained in detail in the following.

Component ProtoMA_i models the protocol part which consists of updating the care-of address at the home agent level. When the mobile node changes its point of attachment, it generates a binding update message, action $generate_{bua}$. Here, the rate associated with this action is unspecified since *ProtoMA* does not generate this message, but has just to transmit it. This is modelled using the firing action $transmit_{bua}$ with rate r_1 . Once component *ProtoMA* is in **HOME_M**, it allows the home agent to update the care-of address using the synchronizing action $updateA$ with rate λ . A *ProtoMA_i*

component is associated with each place \mathbf{DOMAIN}_i , $i = 1, \dots, N$.

$$\begin{aligned} ProtoMA_i &\stackrel{def}{=} (generate_{bua}, \top).(\mathbf{transmit}_{bua}, r_1).ProtoMA_{i1} \\ ProtoMA_{i1} &\stackrel{def}{=} (updateA, \lambda).(\mathbf{return}_i, r_2).ProtoMA_i \end{aligned}$$

Component CommuMA_i allows us to model the exchanges between the home agent and the mobile node once the binding update message has been received by the home agent. Once the communication is established (via *newSessionA*), the home agent will either generate a packet or an update request to be forwarded to the mobile node using firing action $\mathbf{forwardA}_{pkt_i}$ or $\mathbf{forwardA}_{rqt_i}$. Back in place \mathbf{DOMAIN}_i , *CommuMA_i* delivers the packet (*deliverA_{pkt}*) or the request (*deliverA_{rqt}*) to the mobile node. The delivery may fail if the mobile node has already migrated somewhere else or just returned to its home network. Component *CommuMA_i* is associated with place \mathbf{DOMAIN}_i . Formally, the behaviour of this component is as follows.

$$\begin{aligned} CommuMA_i &\stackrel{def}{=} (newSessionA, \top).(\mathbf{openA}, \alpha_1).CommuMA_{i0} \\ CommuMA_{i0} &\stackrel{def}{=} (generateA_{pkt}, \top).(\mathbf{forwardA}_{pkt_i}, \alpha_3).CommuMA_{i2} \\ &\quad + (generateA_{rqt}, \top).(\mathbf{forwardA}_{rqt_i}, \alpha_5).CommuMA_{i4} \\ &\quad + (\mathbf{close}_i, \alpha_2).CommuMA_i \\ CommuMA_{i2} &\stackrel{def}{=} (deliverA_{pkt}, \mu_1).CommuMA_{i3} + (fail, \mu_2).CommuMA_{i3} \\ CommuMA_{i3} &\stackrel{def}{=} (\mathbf{return}, \alpha_4).CommuMA_{i0} \\ CommuMA_{i4} &\stackrel{def}{=} (deliverA_{rqt}, \mu_3).CommuMA_{i3} + (fail, \mu_4).CommuMA_{i3} \end{aligned}$$

Component ProtoMC_i models the protocol part which consists of updating the care-of address at the correspondent level. Once the binding update message is generated, it is forwarded to the correspondent using firing activity $\mathbf{transmit}_{buc}$. Then it allows the home agent to update the care-of address using the synchronizing action *updateC* with rate λ_1 .

$$\begin{aligned} ProtoMC_i &\stackrel{def}{=} (generate_{buc}, \top).(\mathbf{transmit}_{buc}, r_3).ProtoMC_{i1} \\ ProtoMC_{i1} &\stackrel{def}{=} (updateC, \lambda_1).(\mathbf{return}_i, r_4).ProtoMC_i \end{aligned}$$

Component CommuMC_i allows us to model the effective exchanges between the mobile node and the correspondent. Once the communication is established (*newSessionC*) by the mobile node, the correspondent may either generate a packet or an update request that will be forwarded using firing transition $\mathbf{forwardC}_{pkt_i}$ or $\mathbf{forwardC}_{rqt_i}$ respectively. The delivery may succeed or fail if the mobile has already moved. In the first case, the mobile node may either stay silent (*silent*) or generate a packet (*generateM_{pkt}*) which is forwarded, using firing transition $\mathbf{forwardM}_{pkt}$, to the correspondent. The communication is considered finished when transition \mathbf{close}_i is fired at the correspondent level. A component *CommuMC_i* is associated with each place

DOMAIN_{*i*}, *i* = 1, ..., *N*.

$$\begin{aligned}
CommuMC_i &\stackrel{def}{=} (newSessionC, \top).(\mathbf{openC}, \beta_1).CommuMC_{i0} \\
CommuMC_{i0} &\stackrel{def}{=} (generateCM_{pkt}, \top).(\mathbf{forwardC}_{pkt-i}, \beta_3).CommuMC_{i1} \\
&\quad + (generateCM_{rqt}, \top).(\mathbf{forwardC}_{rqt-i}, \beta_5).CommuMC_{i4} \\
&\quad + (\mathbf{close}_i, \beta_2).CommuMC_i \\
CommuMC_{i1} &\stackrel{def}{=} (deliverC_{pkt}, v_1).CommuMC_{i2} + (fail, v_2).CommuMC_{i5} \\
CommuMC_{i2} &\stackrel{def}{=} (generateM_{pkt}, \top).(\mathbf{forwardM}_{pkt}, \beta_4).CommuMC_{i3} \\
&\quad + (silent, v_3).CommuMC_{i5} \\
CommuMC_{i3} &\stackrel{def}{=} (deliverM_{pkt}, v_4).CommuMC_{i0} \\
CommuMC_{i4} &\stackrel{def}{=} (deliverC_{rqt}, v_5).CommuMC_{i5} + (fail, v_6).CommuMC_{i5} \\
CommuMC_{i5} &\stackrel{def}{=} (\mathbf{return}, \beta_6).CommuMC_{i0}
\end{aligned}$$

Component ProtoAC models the case where the mobile node returns to its home network and has to send a binding update message to its current correspondent. This component is associated with place **HOME_M** and returns to it once the care-of address has been updated at the correspondent level using activity *updateCh* with rate λ_2 .

$$\begin{aligned}
ProtoAC &\stackrel{def}{=} (generateH_{buc}, \top).(\mathbf{transmit}_{buc}, r_5).ProtoAC_1 \\
ProtoAC_1 &\stackrel{def}{=} (updateCh, \lambda_2).(\mathbf{return}, r_6).ProtoAC
\end{aligned}$$

Component CommuAC models the communication between the correspondent and the home agent if the mobile node has left its home network or with the mobile node itself if not. It is associated with place **HOME_C** and forwards the packets generated by the correspondent to the mobile node in its home network. This is modelled using the firing action $\mathbf{forwardC}_{pkt}$ with rate γ_1 . If the mobile node is present, the packets are delivered to it with action *deliverC_{pkt}*. Component *CommuAC* then forwards the packets generated by the mobile node to the correspondent. This is done with the firing action $\mathbf{forwardM}_{pkt}$ with rate γ_2 . If the mobile node is not in its home network, the correspondent's packets are saved (*saveC_{pkt}*) by the home agent and component *CommuAC* goes back to **HOME_C** using firing action \mathbf{return} at rate γ_3 .

$$\begin{aligned}
CommuAC &\stackrel{def}{=} (generateCA_{pkt}, \top).(\mathbf{forwardC}_{pkt}, \gamma_1).CommuAC_1 \\
CommuAC_1 &\stackrel{def}{=} (deliverC_{pkt}, w_1).CommuAC_2 + (saveC_{pkt}, w_2).CommuAC_4 \\
CommuAC_2 &\stackrel{def}{=} (generateM_{pkt}, w_3).(\mathbf{forwardM}_{pkt}, \gamma_2).CommuAC_3 \\
&\quad + (silent, w_4).CommuAC_4 \\
CommuAC_3 &\stackrel{def}{=} (deliverM_{h_{pkt}}, w_5).CommuAC \\
CommuAC_4 &\stackrel{def}{=} (\mathbf{return}, \gamma_3).CommuAC
\end{aligned}$$

Component Mobile models the behaviour of the mobile node whatever its current point of attachment. It may generate packets, receive packets or simply choose to move to another network, modelled using actions *generateM_{pkt}*, *deliverC_{pkt}* and $\mathbf{migrate}_i$ respectively. When the mobile node changes its point of attachment, it first generates a binding update message for its home agent with action *generate_{bua}* at rate τ_2 and then opens a new communication session (*newSessionA*). It may then receive from

its home agent either an update request ($deliverA_{rqt}$) or a correspondent's packet ($deliverA_{pkt}$). In this last case, it generates a binding update message for the correspondent with action $generate_{buc}$ at rate τ_3 and establishes a new communication session ($newSessionC$). In network i , the mobile node may stay in the current network or, with probability p_j , move to another network j with firing action $\mathbf{migrate}_j$, $j \neq i$. It may also return to its home network with the firing action \mathbf{return}_h .

$$\begin{aligned}
Mobile &\stackrel{def}{=} (generateM_{pkt}, \tau_1).Mobile + (deliverC_{pkt}, \top).Mobile \\
&\quad + \sum_{i=1}^N (\mathbf{migrate}_i, p_i \times \delta_1).Mobile_1 \\
Mobile_1 &\stackrel{def}{=} (generate_{bua}, \tau_2).(newSessionA, s_1).Mobile_2 \\
Mobile_2 &\stackrel{def}{=} (deliverA_{pkt}, \top).Mobile_3 + (deliverA_{rqt}, \top).Mobile_1 \\
&\quad + (\mathbf{return}_h, \delta_2).Mobile \\
Mobile_3 &\stackrel{def}{=} (generate_{buc}, \tau_3).(newSessionC, s_2).Mobile_4 \\
Mobile_4 &\stackrel{def}{=} (deliverA_{rqt}, \top).Mobile_7 + (deliverA_{pkt}, \top).Mobile_4 \\
&\quad + (deliverC_{rqt}, \top).Mobile_8 + (deliverC_{pkt}, \top).Mobile_4 \\
&\quad + (generateM_{pkt}, \tau_4).Mobile_4 + (\mathbf{return}_h, \delta_2).Mobile_5 \\
&\quad + \sum_{j=1/j \neq i}^N (\mathbf{migrate}_j, p_j \times \delta_5).Mobile_6 \\
Mobile_5 &\stackrel{def}{=} (generate_{buc}, \tau_5).Mobile \\
Mobile_6 &\stackrel{def}{=} (generate_{bua}, \tau_6).(newSessionA, s_1).Mobile_3 \\
Mobile_7 &\stackrel{def}{=} (generate_{bua}, \tau_7).Mobile_4 \\
Mobile_8 &\stackrel{def}{=} (generate_{buc}, \tau_3).Mobile_4
\end{aligned}$$

Component Agent models the home agent's behaviour. It is given by the following PEPA equations:

$$\begin{aligned}
Agent &\stackrel{def}{=} (updateA, \top).Agent_1 + (saveC_{pkt}, \top).Agent_3 \\
Agent_1 &\stackrel{def}{=} (saveC_{pkt}, \top).Agent_2 + (generateA_{rqt}, \nu_1).Agent_1 + (updateA, \top).Agent_1 \\
Agent_2 &\stackrel{def}{=} (generateA_{pkt}, \nu_2).Agent_1 + (updateA, \top).Agent_2 + (saveC_{pkt}, \top).Agent_2 \\
Agent_3 &\stackrel{def}{=} (saveC_{pkt}, \top).Agent_3 + (updateA, \top).Agent_2
\end{aligned}$$

Component Corresp models the behaviour of a correspondent of the mobile node. It may generate packets to send to the mobile node in its home network ($generateCA_{pkt}$) and receive packets from the mobile node when it is still there ($deliverMh_{pkt}$). It may also receive a binding update message from the mobile node ($updateC$). In this case, the correspondent may generate packets ($generateCM_{pkt}$) or update requests ($generateCA_{rqt}$) to send to the mobile node in its current attachment point. Action type $updateCh$ models the case where the correspondent receives a binding update message from the mobile node back in its home network.

$$\begin{aligned}
Corresp &\stackrel{def}{=} (generateCA_{pkt}, c_1).Corresp + (deliverMh_{pkt}, \top).Corresp \\
&\quad + (updateC, \top).Corresp_1 \\
Corresp_1 &\stackrel{def}{=} (generateCM_{pkt}, c_2).Corresp_1 + (generateCA_{rqt}, c_3).Corresp_1 \\
&\quad + (deliverM_{pkt}, \top).Corresp_1 + (updateC, \top).Corresp_1 \\
&\quad + (updateCh, \top).Corresp
\end{aligned}$$

The Places The places of the PEPA net are defined as follows:

$$\begin{aligned}
HOME_M &\stackrel{def}{=} \left(ProtoAC[ProtoAC] \bowtie_{L_1} \left(Mobile[Mobile] \bowtie_{L_2} \left(CommuAC[-] \bowtie_{L_3} \right. \right. \right. \\
&\quad \left. \left. \left(CommuMA_1[-] \parallel \dots \parallel CommuMA_N[-] \right) \bowtie_{L_4} \left(Agent \bowtie_{L_5} \right. \right. \right. \\
&\quad \left. \left. \left(ProtoMA_1[-] \parallel \dots \parallel ProtoMA_N[-] \right) \right) \right) \\
HOME_C &\stackrel{def}{=} \left(\left(\left(ProtoAC[-] \bowtie_{L_6} Corresp \right) \bowtie_{L_7} \left(CommuMC_1[-] \parallel \dots \parallel CommuMC_N[-] \right) \right) \right. \\
&\quad \left. \bowtie_{L_8} \left(ProtoMC_1[-] \parallel \dots \parallel ProtoMC_N[-] \right) \right) \bowtie_{L_9} CommuAC[CommuAC] \\
DOMAIN_i &\stackrel{def}{=} \left(\left(\left(ProtoMA_i[ProtoMA_i] \bowtie_{L_{10}} Mobile[-] \right) \bowtie_{L_{11}} CommuMA_i[CommuMA_i] \right) \right. \\
&\quad \left. \bowtie_{L_{12}} ProtoMC_i[ProtoMC_i] \right) \bowtie_{L_{13}} CommuMC_i[CommuMC_i]
\end{aligned}$$

where $i = 1 \dots N$ and the synchronizing sets are defined as follows

$$\begin{aligned}
L_1 &= \{generateH_{buc}\} & L_8 &= \{updateC\} \\
L_2 &= \{deliverC_{pkt}, generateM_{pkt}\} & L_9 &= \{generateCA_{pkt}, deliverMh_{pkt}\} \\
L_3 &= \{saveC_{pkt}\} & L_{10} &= \{generate_{buc}\} \\
L_4 &= \{generateA_{pkt}, generateA_{rqt}\} & L_{11} &= \{newSessionA, deliverA_{pkt}, deliverA_{rqt}\} \\
L_5 &= \{mathitupdateA\} & L_{12} &= \{generate_{buc}\} \\
L_6 &= \{updateCh\} & L_{13} &= \{newSessionC, generateM_{pkt}, deliverC_{pkt}, deliverC_{rqt}\} \\
& & L_7 &= \{deliverM_{pkt}, generateCM_{pkt}, generateCM_{rqt}\}
\end{aligned}$$

5 Tool Support

The PEPA stochastic process algebra is supported by a range of tools including the PEPA Workbench [12] and the Möbius Modelling Framework [9]. We have implemented the PEPA nets formalism as an extension of the PEPA Workbench. The PEPA modelling tools, together with user documentation, papers and examples are available from the PEPA Web page which is located at <http://www.dcs.ed.ac.uk/pepa>.

We have provided tool support for PEPA nets in two forms. The PEPA Workbench for PEPA nets is a dedicated tool which can be used to generate the Markov process underlying a PEPA net in a format suitable for solution by a number of solvers. This has recently been enhanced by the addition of a graphical editor based on DRAWNET [?]. In contrast, the PEPA net compiler allows the existing tool support for PEPA to be exploited by translating a PEPA net model into an equivalent PEPA model [14]. For example, this gives us the ability to use the PRISM probabilistic symbolic model checker [22] which has been extended to support PEPA.

The use of the PEPA Workbench for PEPA nets is illustrated in Figure 7. *The content of the figure should be changed to be related to one of the examples presented in this paper.* The input language of 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.

6 Related work

Stochastic Petri nets and stochastic process algebras have complementary strengths. A comparison of the two formalisms [10] concludes that “there is scope for future work incorporating the attractive characteristics of the formalisms ... from one paradigm into the other”. Some work has been done in this area in beginning to develop a structural theory for process algebras [15] on the one hand and in importing composition operations from stochastic process algebras into net formalisms on the other [28, 19, 17]. In contrast the work on PEPA nets aims to use both Petri nets and process algebras together as a single, structured performance modelling formalism.

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 [27]) and queueing models (used with generalised stochastic Petri nets in [3]). 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* [2] and Haverkort’s *Dynamic Queueing Networks* [16]. However, these proposals differ from PEPA nets in that the two formalisms may be regarded as adjuncts to one another, with one providing delays to the other, rather than integrated into a single formalism.

Somewhat closer to our own work is Valk’s work on *Elementary Object Systems* [29]. In this work an extension of Petri nets is presented in which the tokens circulating in the net structure (called the *System net*) are themselves Petri nets (termed *Object nets*). Object nets move like ordinary tokens and they can change their markings but not their structure. Three different types of transitions are defined. Transitions occurring in the Object net (i.e. in the marking) are called *system autonomous* and represent the object internal behaviour. An *interaction* takes places when both the Object and the System net enable transitions with the same attached label. A third type of transition causes a change in the System net only and it is called *transport*. In PEPA nets we do not allow such transitions, since a firing cannot occur without modifying the state of a component.

Despite the superficial similarities there are some quite strong differences between the work on PEPA nets and that on Elementary Object Systems (EOS). Fundamentally,

```
PEPA Workbench for PEPA Nets Version 0.81.1 "Granby Road"
{[ Setting model aggregation on ]}
{[ Use of priorities is enabled ]}
Compiling the model
Generating the derivation graph
The model has 49416 states
The model has 147274 transitions
The model has 571362 firings
Writing the hash table file to model3.hash
Exiting PEPA Workbench.
```

Fig. 7. The PEPA Workbench for PEPA nets processing the hierarchical cellular network example for seven clients

EOS are without any timing considerations, other than the relative timing imposed by the Petri net causality relation. In PEPA nets, in addition to this implicit timing information we have explicit time delays integrated into behaviour at both the net level and the token level. Moreover the origins of the works are distinct. Valk's work is motivated by a desire to provide a fundamental model of object-oriented programming, and the development of EOS has been strongly influenced by this goal. Our motivation has been to develop a convenient high-level modelling language for Markov processes, for systems exhibiting mobility. Recent work by Köhler *et al.* has examined the possibility of using Object Systems for modelling mobility and mobile agents although it is not clear what the objective of their modelling is [21].

Several process calculi have been developed specifically for modelling mobile computation, primarily for the purpose of functional verification, the most notable being the π -calculus [25] and the calculus of mobile ambients [7]. The π -calculus, and Priami's subsequent extension, the stochastic π -calculus [26], have a very different style of representing systems [4], which does not satisfy our criterion of clearly separating state changes into distinct types related, in the case of mobile computation, to concepts of location and mobility. In this respect our formalism is closer to the work on mobile ambients.

The calculus of mobile ambients is intended to capture notion of *locations*, *mobility* and *authority for movement*. This is achieved by introducing the concept of *ambient*, i.e. a bounded place where computation happens. Ambients can be nested into other ambients and can be moved as a whole. Mobility primitives are provided by considering *capabilities*: it is possible to *enter* into another ambient, to *exit* from an ambient, to *open* an ambient. Processes are executed within ambients and a simple asynchronous communication mechanism that works within a single ambient is chosen. Communication across ambients is modelled as the movement of 'messenger' agents that must cross ambient boundaries. (This is similar in style to our own representation of the Mobile IP protocol.) The most pronounced differences between PEPA nets and the ambient calculus are the lack of timing information in the ambient calculus and the ability to nest ambients which gives a hierarchical structure to locations which cannot be matched by the places in PEPA nets.

In the performance arena our work has some resonances with earlier work in which Buchholz considered solution techniques for Markov processes which were specified in an hierarchical fashion [5, 6]. A number of component Markov processes were connected via a higher-level model. The higher-level model determined how entities move between lower-level models. The lower-level models in Buchholz's work were principally intended to be queueing networks, so that the entities (customers) which move between lower-level models are themselves without state and have no power to evolve independently.

7 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 formal-

ism 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.

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.

Acknowledgements

Stephen Gilmore, Jane Hillston and Leïla Kloul are supported by the DEGAS (Design Environments for Global ApplicationS) IST-2001-32072 project funded by the FET Proactive Initiative on Global Computing.

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. 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.
3. 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 14th UK Performance Engineering Workshop.
4. L. Brodo, S. Gilmore, J. Hillston, and C. Priami. A stochastic π -calculus semantics for PEPA nets. In *Proc. of the Workshop on Process Algebras and Stochastically Timed Activities*, pages 1–17. LFCS, University of Edinburgh, June 2002.
5. P. Buchholz. Hierarchical Markov Models — symmetries and aggregation. *Performance Evaluation*, 22:93–110, 1995.
6. P. Buchholz. Multi-level solutions for structured Markov chains. *SIAM Journal on Matrix Analysis and Applications*, 22(2):342–357, 2000.
7. L. Cardelli and A.D. Gordon. Mobile ambients. In M. Nivat, editor, *Foundations of Software Science and Computational Structures*, volume 1378 of *LNCS*, pages 140–155. Springer Verlag, 1998.
8. G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models*. PhD thesis, The University of Edinburgh, 2000.

9. G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proc. of 9th Int. Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001.
10. S. Donatelli, J. Hillston, and M. Ribaud. 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.
11. A. Fuggetta, G.P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, may 1998.
12. 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.
13. S. Gilmore, J. Hillston, L. Kloul, and M. Ribaud. PEPA nets: A structured performance modelling formalism. *Performance Evaluation*, 2(2):1–10, 2003.
14. S. Gilmore, J. Hillston, L. Kloul, and M. Ribaud. Software performance modelling using PEPA nets. to appear in proc. of Int. Workshop on Software Performance, 2004.
15. 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.
16. B.R. Haverkort, I.G. Niemegeers, and P. Veldhuyzen van Zanten. DyQNtool—a performativity 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.
17. H. Hermanns, U. Herzog, V. Mertsiotakis, and M. Rettelbach. Exploiting stochastic process algebra achievements for generalized stochastic Petri nets. In *Proc. of 7th International Workshop on Petri Nets and Performance Models*, Saint Malo, June 1997. IEEE CS Press.
18. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
19. R. Hopkins and P. King. A visual formalism for the composition of stochastic Petri nets. In T. Field *et al.*, editor, *Proc. of 12th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 2324 of LNCS, pages 239–258, London, 2002. Springer.
20. D. Johnson, C. Perkins, and J. Arkko. Mobility support in IPv6. IETF Mobile IP Working Group Internet-Draft draft-ietf-mobileip-ipv6-20, January 2003.
21. M. Köhler, D. Moldt, and H. Rölke. Modelling mobility and mobile agents using nets within nets. In W.M.P. van der Aalst and E. Best, editors, *Proc. of Int. Conf. on Applications and Theory of Petri Nets*, volume 2679 of LNCS, pages 121–139. Springer-Verlag, June 2003.
22. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field *et al.*, editor, *Proc. of 12th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 2324 of LNCS, London, 2002. Springer.
23. K. Larsen and A. Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94(1):1–28, September 1991.
24. 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.
25. R. Milner. *Communicating and Mobile Systems: the π -calculus*. Cambridge University Press, 1999.
26. 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.

27. 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.
28. I. C. Rojas M. *Compositional Construction and Analysis of Petri net Systems*. PhD thesis, The University of Edinburgh, 1997.
29. R. Valk. Petri nets as token objects—an introduction to Elementary Object Nets. In J. Desel and M. Silva, editors, *Proc. of 19th Int. Conf. on Application and Theory of Petri Nets*, volume 1420 of *LNCS*, pages 1–25, Lisbon, Portugal, 1998. Springer-Verlag.

A Semantics of PEPA

The semantic rules, in the structured operational style, are presented in Figure 8; the interested reader is referred to [18] 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 .

A.1 Additional semantic rules for PEPA nets

For any token component its action type set can be partitioned in distinct subsets corresponding to transitions and firings respectively. Thus for a component Q , $\mathcal{A}_t(Q)$ is the set of local transitions currently enabled in Q and $\mathcal{A}_f(Q)$ is the set of firings currently enabled for Q .

We use the notation

$$\mathbf{P}_1 \xrightarrow{(\alpha, r)} \mathbf{P}_2$$

to capture the information that there is a transition connecting place \mathbf{P}_1 to place \mathbf{P}_2 labelled by (α, r) . This relation captures static information about the structure of the net, not dynamic information about its behaviour. The semantic rules for PEPA nets are those for PEPA plus the additional rules presented in Figure 9;

The Cell rule conservatively extends the PEPA semantics to define that a cell which is filled by a component Q has the same transitions as Q itself. A healthiness condition on the rule (a *typing judgement*) requires a context $Q[-]$ to be filled with a component which has the same alphabet as Q . We write $Q =_a Q'$ to state that Q and Q' 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). Recall that the transition and firing alphabets are distinct. We do not give priority to one alphabet of actions over the other; the highest-priority firings and the transitions compete based on a race policy.

The Firing rule takes one marking of the net to another 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. In order to take account of the priorities we define a number of supplementary transition relations,

Prefix

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

Cooperation

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

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

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

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

Constant

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

Fig. 8. The operational semantics of PEPA

one for each priority level. A net level transition's eligibility for firing depends on two conditions. Firstly there must be an empty cell in the destination place into which the token can be transferred. The Enabling rule ensures that this is the case, and defines a transition relation, decorated with the priority level of the corresponding activity type. The rate at which the activity is enabled is calculated as in the PEPA semantics of cooperation. In order for a firing to take place it must also be the case that the type of the enabled firing has the highest priority level in the set of the enabled firings. This is imposed by the Firing rule in which we discard those enabled firings which do not have the highest priority. In other words for a firing to occur there must not be any other firing satisfying the Enabling rule (empty destination cell) which has a higher priority.

Cell:

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

Transition:

$$\frac{M_{\mathbf{P}} \xrightarrow{(\alpha, r)} M'_{\mathbf{P}}}{(\dots, M_{\mathbf{P}}, \dots) \xrightarrow{(\alpha, r)} (\dots, M'_{\mathbf{P}}, \dots)} \quad (\alpha \in \mathcal{A}_t)$$

Enabling:

$$\frac{Q \xrightarrow{(\alpha, r_1)} Q' \quad \mathbf{P}_i \xrightarrow{(\alpha, r_2)} \mathbf{P}_j}{(\dots, \mathbf{P}_i[\dots, Q, \dots], \dots, \mathbf{P}_j[\dots, -, \dots], \dots) \xrightarrow{\pi(\alpha)} (\dots, \mathbf{P}_i[\dots, -, \dots], \dots, \mathbf{P}_j[\dots, Q', \dots], \dots)} \quad (\alpha \in \mathcal{A}_f)$$

Firing:

$$\frac{M \xrightarrow{(\alpha, r)}_n M' \quad M \xrightarrow{(\beta, s)}_m M''}{M \xrightarrow{(\alpha, r)} M'} \quad (n \geq m)$$

Fig. 9. Additional semantic rules for PEPA nets

A.2 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.

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

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

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

