Performance Evaluation for Global Computation

Linda Brodo[1], Pierpaolo Degano[2], Stephen Gilmore[3], Jane Hillston[3], and Corrado Priami[4]

[1] Istituto Trentino Cultura – IRST
Via Sommarive 18, I-38050 Povo (Trento), Italy
brodo@itc.it
[2] Dipartimento di Informatica
Università di Pisa
F.Buonarroti 2, I-56127 Pisa, Italy
degano@di.unipi.it
[3] Laboratory for Foundations of Computer Science
The University of Edinburgh
Edinburgh EH9 3JZ, Scotland
{stg,jeh}@lfcs.ed.ac.uk
[4] Università di Trento
Dipartimento di Informatica e Telecomunicazioni
Via Sommarive 14, I-38050 Povo (Trento), Italy
priami@dit.unitn.it

Abstract. Global computing applications co-ordinate distributed computations across widely-dispersed hosts. Such systems present formidable design and implementation challenges to software developers and synchronisation, scheduling and performance problems come to the fore. Complex systems such as these can benefit from the application of high-level performance analysis methods founded on timed process algebras. In this paper we compare the use of two such approaches, the PEPA nets and EOS methods, illustrating our presentation with the example of modelling Web services.

1 Introduction

Our main concern here is comparing existing process algebraic primitives against the needs arising when modelling global applications with a view to determining their run-time performance. Communication and especially mobility are possibly the two main features characterising global computing. There are different approaches to their representation. One widely-studied approach represents mobility implicitly through the communication of links. A name n, representing a communication channel, is passed to an agent that now becomes connected through n to all the agents that know the link n. In this way the topology of the interconnecting network varies while the (distributed) computation goes on. The typical representative of this class is the well-known π-calculus [16].

An alternative approach to representing mobility and communication is taken by the PEPA nets formalism, which combines the process algebra PEPA with

C. Priami (Ed.): GC 2003, LNCS 2874, pp. 229–253, 2003.
© Springer-Verlag Berlin Heidelberg 2003

a Petri net infrastructure [9]. In this formalism, which can be regarded as a high-level Petri net formalism, *places* are process algebra contexts and *tokens* are process algebra components. Mobility is modelled explicitly by the firing of a transition in the Petri net which has the result of a component moving from one place to another. Communication is restricted to be local and is modelled by the usual process algebra communication between components. We have previously studied the relationship between PEPA nets and the π-calculus, by translating a subset of the PEPA nets formalism into the stochastic π-calculus [2]. The objective of performance evaluation is to analyse the dynamic behaviour of a system and predict performance *indices* or *measures* such as throughput, utilisation or response time. A performance prediction may be useful at specification time both when the system implementation is known and when the specification is not connected to any particular implementation. In the first case analysis can suggest potential technical improvements of the implementation. In the second case many implementations can be imagined for the same specification and this could give hints in choosing the more adequate implementation. If a process algebra model is to be used for this purpose certain aspects of the behaviour of the model must be quantified. For example, in classical process algebras alternative behaviours are modelled by a non-deterministic choice. However from such a model no predictions about the likelihood of differing behaviours can be made. Therefore when the objective is performance evaluation, non-deterministic choice is replaced by probabilistic choice. Similarly, in the dynamic behaviour of a system the durations of actions (or equivalently the delays between events) are important and must be incorporated into the model.

Many probabilistic and timed extensions of process algebras have appeared in the literature in the last 15 years, however the most prevalent approach taken in performance evaluation is exemplified by PEPA [11]. In this language all actions have an associated duration which is specified by a random variable, governed by a negative exponential distribution. In PEPA probabilistic choice is not modelled explicitly; when more than one activity is possible it is assumed that the activities race in the sense that each draws from the corresponding distribution function to obtain a duration for this instance of the activity. The activity with the shorter duration sample is the one which will be performed first, thus “winning the race”. In practice, since all durations are governed by a negative exponential distributions, the relative probability of activities in competition can be derived by a simple formula. The stochastic π-calculus [18] adopts similar constructs.

In this way, our process algebra models can be used to generate a Continuous Time Markov Chain (CTMC) which can be solved to obtain a steady state probability distribution from which performance measures can be derived. In recent work we have extended PEPA to allow the durations of activities to be defined via functions rather than explicitly [13, 12]. In the context of global computing this means that the duration of an activity can depend on the state of other components. In PEPA nets, again the target representation for performance analysis is a CTMC, and so both process algebra transitions, and Petri net firings have an associated duration which is negative exponentially distributed. As previously conflicts may be solved by the race policy but it is also possible to assign different priorities to different Petri net transitions, giving some firings priority over others [9].

In the EOS approach [6], the transition labels are enhanced so that they record the application of inference rules. The designer of the application under analysis can define evaluation functions that determine the rates of transitions, by inspecting enhanced labels, as these represent the low level routines performed by the run-time support to execute the transition itself. Since the rates are also affected by the target architecture, its peculiarities will also affect the evaluation functions, the parameters of which are then the enhanced labels and the architectural details [17].

Structure of this paper: The paper is organized as follows. In the next section we describe our running example of Web services. Section 3 recalls the basics of PEPA nets, while Sect. 4 introduces the PEPA nets semantics and shows how performance analysis can be carried out on the running example. Section 5 describes the EOS approach on the π-calculus and shows how it can be used to perform a quantitative analysis of the Web service system. Finally, we draw some conclusions.

2 Example: Modelling Web Services

Web services provide a technological platform which enables global computation. In a Web services architecture clients and services are loosely coupled and geographically distributed. Services are obtained by discovery from registries and directories. Web service descriptions specify the interfaces and locations of services. Method invocation and transport of data are performed by asynchronous message passing. The computing platform is heterogeneous and architecture-neutral. The implementation platform is also heterogeneous; web service clients may be implemented in a different implementation language from the service application. All of the above qualities typify global computation: distributed computations across heterogenous platforms utilising discovery services to effect remote evaluation.

Another typical quality of global computations is that they take place across administrative domains. In consequence they must coordinate communication and evaluation across different security contexts. Firewalls are used in distributed systems to safeguard systems against attack, preventing unrestricted communication between remote sites. Their presence is a necessity but one which causes problems for some communications protocols. Web services however are accessed by HTTP. The use of the HTTP protocol virtually eliminates the complications caused by firewalls.

Web services achieve global accessibility in practice by adherance to open standards which are widely supported and used. Both communication protocols and data formats are standardised. Web services are globally positioned by giving each a unique Uniform Resource Identifier (URI). Clients and services in a Web services architecture exchange XML-encoded messages using the standard SOAP protocol. The use of XML for data carriage provides an abstraction barrier over the language-dependent in-memory data formats used in application programs. The SOAP protocol provides a high-level transport and may itself be layered over native network protocols such as SMTP or HTTP. A special-purpose language WSDL (Web Services Description Language) exists for describing interface "contracts" between Web service provider and client.

Web services applications incorporate many significant practical advances over previous generations of distributed systems technology. One cost of their considerable advantages is that they are resource-intensive systems. Web services include many layers of encapsulation which would not be needed in traditional binary communication protocols. Service lookup is an overhead, as is XML-encoding. The XML language itself is a verbose, human-readable encoding format which is engineered for clarity, not for compactness. This has the consequence that XML-encoded method calls are weighty data items which incur significant transmission costs. The use of the HTTP protocol is another overhead. Network reliability, host availability problems and distributed system faults further degrade performance. For these reasons, Web services provide a highly appropriate example for performance modelling techniques such as those presented in this paper.

Process algebras are excellent tools for modelling Web services because they naturally support peer-to-peer architectures. The co-operator/co-operand style of process algebras allows an intuitive encoding of control flow logics such as callbacks. A process algebra which provides direct support for location-awareness is an added benefit. This provides the right conceptual modelling concepts to represent mobile code systems ranging from the asynchronous remote procedure call method provided by Web services to more complex configurations as embodied in the remote evaluation, code-on-demand or mobile agent paradigms.

3 PEPA Nets

PEPA nets extend the PEPA [11] stochastic process algebra by connecting individual PEPA models together as the places of a coloured stochastic Petri net. PEPA components travel from place to place as the tokens of the net.

A PEPA net differentiates between two types of change of state. We refer to these as *firings* of the net and *transitions* of PEPA components. Each are special cases of PEPA activities. Transitions of PEPA components will typically be used to model small-scale changes of state as components undertake activities. Firings of the net will typically be used to model large-scale 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.

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. Maintaining this strict distinction between firings and transitions is essential in order to provide the separation into macro- and micro-step state changes that we are seeking to represent.

Each place has a distinct alphabet for transitions and firings, meaning that the same action type cannot be used for both. Thus there can be no ambiguity between such micro- and macro-scale transitions.

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. The components which fill cells can circulate as the tokens of the net. In contrast, the static components cannot move.

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'. $Q[Q'']$ enables the same transitions as Q''. 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. This has as a consequence that if Q' is a derivative of Q then it is also the case that Q is a derivative of Q', for any Q and Q'.

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

For any token component its action type set can be partitioned in distinct subsets corresponding to transitions and firings respectively. For a component Q we will denote these sets by $\mathcal{A}_t(Q)$ and $\mathcal{A}_f(Q)$, where $\mathcal{A}_t(Q)$ is the set of local transitions currently enabled in Q and $\mathcal{A}_f(Q)$ is the set of firings currently enabled for Q. Note that for a firing to be enabled the token must enable the corresponding activity, it must be in a place connected to a net-level transition

Table 1. The syntax of PEPA extended with contexts.

$N ::= D^{+}M$	(net)
(definitions and marking)	

$M ::= (M_{\mathbf{P}}, \ldots)$	(marking)	$D ::= I \stackrel{def}{=} S$	(component defn)
$M_{\mathbf{P}} ::= \mathbf{P}[C, \ldots]$	(place marking)	$\mid\ \mathbf{P}[C] \stackrel{def}{=} P[C]$	(place defn)
		$\mid\ \mathbf{P}[C, \ldots] \stackrel{def}{=} P[C] \bowtie_{L} P$	(place defn)
(marking vectors)		(identifier declarations)	

$S ::= (\alpha, r).S$	(prefix)	$P ::= P \bowtie_{L} P$	(cooperation)	$C ::=$ '_'	(empty)
$\mid\ S + S$	(choice)	$\mid\ P/L$	(hiding)	$\mid\ S$	(full)
$\mid\ I$	(identifier)	$\mid\ P[C]$	(cell)		
		$\mid\ I$	(identifier)		
(sequential components)		(concurrent components)		(cell term expressions)	

of the same type and there must be an empty cell at the output place of the transition of the correct token type.

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 **a** and **b**).

3.1 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 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, $(2, 1, 0)$.

For a PEPA net, we can denote a marking by $(\mathbf{P_1}[Q], \mathbf{P_2}[\_], \mathbf{P_3}[\_])$ (the token at place $\mathbf{P_1}$ is in state Q; the other places have no tokens). In general, a context may have more than one parameter, to be filled by PEPA components of different types. We denote the ith component of a marking M by M_i. For example, $(\mathbf{P_1}[Q], \mathbf{P_2}[\_], \mathbf{P_3}[\_])_1$ is $\mathbf{P_1}[Q]$.

It is simple to define a function to count the number of tokens in a PEPA net term and this function proves to be useful in practice.

$$\begin{aligned}
\text{tokens}(P) &= 0 \\
\text{tokens}(P[_]) &= 0 \\
\text{tokens}(P[P']) &= 1 \\
\text{tokens}(P \underset{L}{\bowtie} Q) &= \text{tokens}(P) + \text{tokens}(Q) \\
\text{tokens}(P/L) &= \text{tokens}(P)
\end{aligned}$$

3.2 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 conjunction with the component which is performing it.

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).

3.3 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 place. 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.

4 Semantics

The PEPA language is formally defined by a small-step operational semantics. 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

$$\mathbf{P_1} \xrightarrow{(\alpha, r)} \mathbf{P_2}$$

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. We could describe the net structure in a PEPA net using a list of such declarations but the more familiar graphical presentation of a net presents the same information in a more accessible way.

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*

- *$\mathcal{P}$ is a finite set of places;*
- *$\mathcal{T}$ is a finite set of net transitions;*
- *$I : \mathcal{T} \to \mathcal{P}$ is the input function;*
- *$O : \mathcal{T} \to \mathcal{P}$ is the output function;*
- *$\ell : \mathcal{T} \to (\mathcal{A}_f, \mathbb{R}^+ \cup \{\top\})$ 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;*
- *$\pi : \mathcal{A}_f \to \mathbb{N}$ is the priority function which assigns priorities (represented by natural numbers) to firing action types;*
- *$\mathcal{C} : \mathcal{P} \to P$ 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.*

The semantic rules for PEPA nets are provided in Table 2. 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 (also called a *typing judgement*) requires a context such as $Q[\_]$ 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 of any place must be distinct.

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.

Table 2. Additional semantic rules for PEPA nets.

Cell:

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

Transition:

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

Enabling:

$$\frac{Q \xrightarrow{(\alpha,\, r_1)} Q' \qquad \mathbf{P_i} \xrightarrow{(\boldsymbol{\alpha},\, r_2)} \mathbf{P_j}}{(.., \mathbf{P_i}[.., Q, ..], .., \mathbf{P_j}[.., \_, ..], ..) \xrightarrow{(\boldsymbol{\alpha},\, R)}_{\pi(\alpha)} (.., \mathbf{P_i}[.., \_, ..], .., \mathbf{P_j}[.., Q', ..], ..)} (\alpha \in \mathcal{A}_f)$$

Firing:

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

4.1 The Net Bisimulation Relation

In this section we define a bisimulation relation for PEPA nets 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 [14], based on a conditional transition rate between *markings*, rather than the strong equivalence relation of PEPA which considers the transition rates between components. 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 $E \in M/\mathcal{R}$,*

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

4.2 PEPA Net Model of a Web Service

In modelling our Web services example as a PEPA net we first identify three components: *Client*, *WebService* and *SOAPmessage*. We begin with the simplest of these, the *SOAPmessage*. The lifecycle of this component is that it is built using a message composition API, then launched over the network and then read using an XML parser. This leads to another message which is the continuation of the lifetime of this component. This component plays the role of passive data in our application so in its description it leaves unspecified ($\top$) the rates at which these actions are performed, allowing the cooperating partner in the synchronisation to determine these rates.

$$\begin{aligned} \mathit{SOAPmessage} \stackrel{\mathit{def}}{=} & (\mathit{compose\_message}, \top). \\ & (\mathbf{launch}, \top). \\ & (\mathit{read\_message}, \top).\mathit{SOAPmessage} \end{aligned}$$

A *Client* divides its time between local computation, the details of which we do not model here, and Web services interactions. When the client comes to a phase in its local computation where it realises that it needs to use a Web service it interacts with the discovery service to obtain a specification of the service. It then composes a SOAP message to send to the service. The communication with the remote Web service is asynchronous so the client returns to its local computation, anticipating that a reply will come later. When a message is returned from the service the client will read it and make use of the results in the remainder of its computation.

$$\begin{aligned} \mathit{Client} & \stackrel{\mathit{def}}{=} (\mathit{local\_computing}, r_l).\mathit{Client} \\ & \quad + (\mathit{discover}, r_d).\mathit{Client}_1 \\ \mathit{Client}_1 & \stackrel{\mathit{def}}{=} (\mathit{compose\_message}, r_c^C).\mathit{Client}_2 \\ \mathit{Client}_2 & \stackrel{\mathit{def}}{=} (\mathit{local\_computing}, r_l).\mathit{Client}_2 \\ & \quad + (\mathit{read\_message}, r_r^C).\mathit{Client} \end{aligned}$$

The lifetime of a Web service is modelled as a simple loop. Web services requests are received and read; these lead to the execution of a Web service and the composition of a message to return the results.

$$\begin{aligned} \mathit{WebService} & \stackrel{\mathit{def}}{=} (\mathit{read\_message}, r_r^S).\mathit{WebService}_2 \\ \mathit{WebService}_2 & \stackrel{\mathit{def}}{=} (\mathit{transact\_service}, r_s).\mathit{WebService}_3 \\ \mathit{WebService}_3 & \stackrel{\mathit{def}}{=} (\mathit{compose\_message}, r_c^S).\mathit{WebService} \end{aligned}$$

The places of the net specify that there is a cell (a storage place) for a SOAP message at the client side and at the Web service side. The message synchronises on composition and reading activities.

$$\begin{aligned} & P_1[s] \stackrel{\mathit{def}}{=} \mathit{SOAPmessage}[s] \bowtie_L \mathit{WebService} \\ & P_2[s] \stackrel{\mathit{def}}{=} \mathit{SOAPmessage}[s] \bowtie_L \mathit{Client} \\ & \text{where } L = \{\, \mathit{compose\_message}, \mathit{read\_message} \,\} \end{aligned}$$

The initial marking of the net places a token on the client side, in its initial state: $(P_1[\_], P_2[\mathit{SOAPmessage}])$.

Firing the operational semantics of the example generates the state space depicted in Fig. 1 with the transition system given in Fig. 2. By erasing activity names from the labelled transition system we obtain the CTMC given in Fig. 3.

As a concrete illustration of numerical evaluation take $r_d = r_m = 17.03$, $r_c^C = r_r^S = r_c^S = r_r^C = 3.28$ and $r_s = 1.10$. The value of r_l is immaterial because the self-loops on states which are visible at the process algebra level are not represented at the Markov chain level. In the Markov chain representation we are concerned with balancing flow into a state against flow out of a state, so self-loops have no role.

Denote the infinitesimal generator matrix of the CTMC in Fig 3 by $\boldsymbol{Q}$. As usual, we solve $\boldsymbol{\pi Q} = \mathbf{0}$ subject to $\sum \boldsymbol{\pi} = 1$ giving (0.025, 0.132, 0.025, 0.132, 0.394, 0.132, 0.025, 0.132).

1	$(SOAPmessage[\_] \bowtie_L WebService,$ $SOAPmessage[SOAPmessage] \bowtie_L Client)$
2	$(SOAPmessage[\_] \bowtie_L WebService,$ $SOAPmessage[SOAPmessage] \bowtie_L Client_1)$
3	$(SOAPmessage[\_] \bowtie_L WebService,$ $SOAPmessage[(\mathbf{launch}, \top).(read\_message, \top).SOAPmessage] \bowtie_L Client_2)$
4	$(SOAPmessage[(read\_message, \top).SOAPmessage] \bowtie_L WebService,$ $SOAPmessage[\_] \bowtie_L Client_2)$
5	$(SOAPmessage[SOAPmessage] \bowtie_L WebService_2,$ $SOAPmessage[\_] \bowtie_L Client_2)$
6	$(SOAPmessage[SOAPmessage] \bowtie_L WebService_3,$ $SOAPmessage[\_] \bowtie_L Client_2)$
7	$(SOAPmessage[(\mathbf{launch}, \top).(read\_message, \top).SOAPmessage] \bowtie_L WebService,$ $SOAPmessage[\_] \bowtie_L Client_2)$
8	$(SOAPmessage[\_] \bowtie_L WebService,$ $SOAPmessage[(read\_message, \top).SOAPmessage] \bowtie_L Client_2)$

Fig. 1. Reachable state space of the PEPA nets Web services model shown as the markings of (P_1, P_2).

4.3 Using Logic to Specify Performance Measures

We now explain how to specify performance measures of interest with respect to a PEPA net model by using a probabilistic modal logic. The appropriate logic for PEPA nets is one which can specify performance measures over the places of the net, and has the capability of expressing requirements on tokens in addition to requirements on the transitions and firings of the net.

We introduce the PML_ν logic by means of a two-level grammar which separates the specification of place formulae and token formulae from the specification of transition and firing activities. Behaviour at the transition and firing level is captured by formulae of a sub-logic, PML_μ.

1	$-(local\_computing, r_l) \rightarrow$	1
1	$-(discover, r_d) \rightarrow$	2
2	$-(compose\_message, r_c^C) \rightarrow$	3
3	$-(local\_computing, r_l) \rightarrow$	3
3	$-(\mathbf{launch}, r_m) \rightarrow$	4
4	$-(read\_message, r_r^S) \rightarrow$	5
4	$-(local\_computing, r_l) \rightarrow$	4

5	$-(transact\_service, r_s) \rightarrow$	6
5	$-(local\_computing, r_l) \rightarrow$	5
6	$-(compose\_message, r_c^S) \rightarrow$	7
6	$-(local\_computing, r_l) \rightarrow$	6
7	$-(\mathbf{launch}, r_m) \rightarrow$	8
7	$-(local\_computing, r_l) \rightarrow$	7
8	$-(local\_computing, r_l) \rightarrow$	8
8	$-(read\_message, r_r^C) \rightarrow$	1

Fig. 2. The transition system of the Web services example.

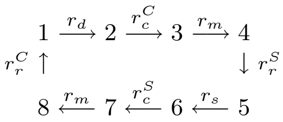


Fig. 3. The CTMC of the Web services example.

This separation of PML_ν formulae from PML_μ formulae enforces a syntactic restricition on the allowable terms in the logic whereby places cannot refer to the local state at another place. This reflects the global computing idiom that it is impossible to know the global state of the system. This restriction also strongly supports the PEPA nets modelling rule which forbids communication between components at different places in the net, as in distributed systems without shared memory.

We present the sub-logic PML_μ first. The constant true is represented by `tt`. Conjunction and negation are denoted as usual. The term ∇_α represents the inability of a process to perform an α action. The diamond operator specifies an activity α, a rate μ, and a succeeding formula which is to be satisfied by all one-step α-derivatives. The accumulated rate of these α activities must be at least μ. We use ϕ, ϕ_1, ϕ_2, ..., to range over PML_μ formulae.

$$
\begin{aligned}
\phi ::= {} & \texttt{tt} \\
\mid {} & \neg\phi \\
\mid {} & \phi_1 \wedge \phi_2 \\
\mid {} & \nabla_\alpha \\
\mid {} & \langle\alpha\rangle_\rho \phi
\end{aligned}
$$

The meaning of the PML_μ connectives is given by reference to the transition relation of the PEPA net semantics. We require an addition simple auxilliary definition:

Definition 1 *Let S be a set of states. $P \overset{(\alpha,\lambda)}{\Longrightarrow} S$ if for all successors $P' \in S$, $P \overset{\alpha}{\longrightarrow} P'$, and $\sum\{\, r : P \overset{(\alpha,r)}{\longrightarrow} P', P' \in S \,\} = \lambda$.*

Now let P be a model of a PEPA net process. Then

$$
\begin{array}{lll}
P \models_\mu \texttt{tt} & & \\
P \models_\mu \neg\phi & \text{iff } P \not\models_\mu \phi & \\
P \models_\mu \phi_1 \wedge \phi_2 & \text{iff } P \models_\mu \phi_1 \wedge P \models_\mu \phi_2 & \\
P \models_\mu \nabla_\alpha & \text{iff } P \stackrel{\alpha}{\not\rightarrow} & \\
P \models_\mu \langle\alpha\rangle_\rho \phi & \text{iff } P \stackrel{(\alpha,\lambda)}{\Longrightarrow} S \text{ for some } \lambda \geq \rho, \text{ and for all } P' \in S, P' \models_\mu \phi.
\end{array}
$$

It is convenient to introduce a number of derived operators. These add no expressive power to the logic but they shorten the statement of realistic performance measures in PML_μ.

$$
\begin{aligned}
\texttt{ff} &\stackrel{def}{=} \neg\texttt{tt} \\
[\alpha]_\rho \phi &\stackrel{def}{=} \neg\langle\alpha\rangle_\rho \neg\phi \\
\Delta_\alpha &\stackrel{def}{=} \neg\nabla_\alpha \\
\phi_1 \vee \phi_2 &\stackrel{def}{=} \neg((\neg\phi_1) \wedge (\neg\phi_2))
\end{aligned}
$$

The PML_ν logic has as atomic propositions all of the formulae of PML_μ. In addition it has conjunction and negation, place formulae and token formulae. We use ψ, ψ_1, ψ_2, ..., to range over PML_ν formulae.

$$
\begin{aligned}
\psi ::= {} & \phi \\
\mid {} & \neg\psi \\
\mid {} & \psi_1 \wedge \psi_2 \\
\mid {} & P_i[\phi] \\
\mid {} & \#P_i \sim n
\end{aligned}
$$

where $\sim = \{=, \neq, <, \leq, >, \geq\}$.

The meaning of PML_ν formulae ($\models_\nu$) is defined in terms of the meaning of PML_μ formulae ($\models_\mu$) and the token counting function for PEPA nets. Let M be a marking of a PEPA net. Then,

$$
\begin{array}{ll}
M \models_\nu \phi & \text{iff } M \models_\mu \phi \\
M \models_\nu \neg\psi & \text{iff } M \not\models_\nu \psi \\
M \models_\nu \psi_1 \wedge \psi_2 & \text{iff } M \models_\nu \psi_1 \wedge M \models_\nu \psi_2 \\
M \models_\nu P_i[\phi] & \text{iff } M_i \models_\mu \phi \\
M \models_\nu \#P_i \sim n & \text{iff } \mathrm{tokens}(M_i) \sim n.
\end{array}
$$

4.4 Selecting States of the Web Services Model

Performance measures characterising the long-run behaviour of the system are calculated from the computation of the probability of being in selected subsets of the states of the system.

We now use PML_ν to characterise some of the states of the Web services PEPA net model, illustrating its use as a specification language for performance measures.

The first value which we might wish to quantify is the *next-read probability.* This is the probability that one of the components of the model can read a message as its next action. We related this formula to the concrete subset of states of the Web services model as shown below:

$$\|\Delta_{read\_message}\| = \{4, 8\}$$

A slightly more specialised quantity is the *server next-read probability*. This is the probability that the Web service component can read a message as its next action. Again we relate a PML_ν formula to a subset of the state space, in this case this just turns out to be just a single state.

$$\|P_1[\Delta_{read\_message}]\| = \{4\}$$

As a final example we can specify the *blocking probability*. This describes the cases where a Web services request message is being processed at the server side and the client is delayed awaiting the reply, performing local computation only. For the present simple example, there are many ways to express this property some of which would also be applicable in a more complex, multi-threaded version of the model. The most direct expressions seem to come from stating the number of tokens at one of the places.

$$\|\#P_1 = 1\| = \|\#P_2 = 0\| = \{4, 5, 6, 7\}$$

5 Enhanced Operational Semantics

In this section we survey Degano and Priami's enhanced operational semantics (EOS for short) [5]. EOS is built upon operational semantics by enriching labels of transitions with the (partial) encodings of their proofs. By exploiting this information, different descriptions of process behaviour can be mechanically derived, thus expressing both quantitative and qualitative aspects [6]. Here, we shall concentrate on a quantitative description that enables us to measure the performance of global applications, specified in the π-calculus [16, 15].

We first recall below the EOS semantics of the π-calculus and then the stochastic interpretation of the enriched labels of the transitions. We shall then consider the web service example introduced in Section 3.

Definition 3. *Let $\mathcal{N}$ be a countable infinite set of* names *which is ranged over by $a, b, \ldots, x, y, \ldots$ with $\tau \notin \mathcal{N}$. We also assume a set $\mathcal{A}$ of agent identifiers ranged over by $A, A_1, \ldots$.* Processes *(denoted by $P, Q, R, \ldots \in \mathcal{P}$) are built from names according to the syntax*

$$P ::= \mathbf{0} \mid \pi.P \mid (\nu x)P \mid P|P \mid P + P \mid A(y_1, \ldots, y_n)$$

where π may be either $x(y)$ for input*, or $\overline{x}\, y$ for* output *(where x is the* subject*, singled out by a function sbj and y the* object*, singled out by a function obj) or τ for* silent *moves. The order of* precedence *among the operators is the order (from left to right) listed above. Hereafter, the trailing $\mathbf{0}$ will be omitted.*

Table 3. Structural congruence for the π-calculus.

$(\nu\, x)(\nu\, x')T \equiv (\nu\, x')(\nu\, x)P$	$A(\tilde{y}) \equiv P,\ if\ A(\tilde{y}) \stackrel{def}{=} P$
$(\nu x)(T_0 \vert T_1) \equiv ((\nu\, x)T_0)\vert T_1,\ \text{if } x \notin fn(T_1)$	$(\nu\, x)\mathbf{0} \equiv \mathbf{0}$

The process $\mathbf{0}$ can perform no actions. The prefix π is the first atomic action that the process $\pi.P$ can perform. The input $x(y)$ binds the occurrences of the variable y in the prefixed process P. Roughly, a name will be received on the channel x and it will substitute the free occurrences of the placeholder y in P. The output prefix $\overline{x}\, z$ sends the name z along the channel x without binding z. In the process $(\nu\, x)P$, the restriction operator $(\nu\, x)$ creates a new (unique) name x whose scope is P. The operator $\vert$ defines the parallel composition of processes. In the composition $P_1 \,\vert\, P_2$ the two processes act independently and they may communicate if they share a common channel name. The summation operator defines the non deterministic choice: $P_1 + P_2$ behaves either as P_1 or as P_2. For each agent identifier A there is a unique defining equation of the form $A(\tilde{y}) \stackrel{def}{=} P$, where $\tilde{y}$ is a list of distinct parameters which are the free names of the process P. Each occurrence of an agent identifier $A(\tilde{z})$ will be replaced by the process P, substituting the list of formal parameters $\tilde{y}$ by the list of actual parameters $\tilde{z}$. Here we assume that the processes associated to agent identifiers contain no parallel operators, *i.e.* have sequential behaviour.

We enrich the labels of transitions with tags that record the rules applied in their derivation and we call the new labels *proof terms*. We also define a function ℓ that maps proofs terms to standard labels.

Definition 4 (proof terms). *Let* $\vartheta \in \{||_0, ||_1, +_0, +_1\}^*$*. Then the set* Θ *of* proof terms (*with metavariable* θ) *is defined by the following syntax*

$$\theta ::= \vartheta\mu \mid \vartheta\langle ||_0\vartheta_0\mu_0, ||_1\vartheta_1\mu_1\rangle$$

with $\mu_i = x(z)$ *iff* μ_{1-i} *is either* $\overline{x}z$ *or* $\overline{x}(z)$*, for* $i \in \{0, 1\}$.

Function $\ell : \Theta \rightarrow Act$ *is defined as*

$$\ell(\theta\mu) = \mu; \quad \ell(\vartheta\langle ||_0\vartheta_0\mu_0, ||_1\vartheta_1\mu_1\rangle) = \tau.$$

Here, we only consider tags that record the occurrences of the parallel and summation operators, as they suffice for the present treatment. A more detailed definition is in [19] that uses tags for all the other π-calculus operators.

The enhanced operational semantics is defined by the inference rules in Tab. 4, assuming the minimal congruence, induced by α-congruence and by the rules in Tab. 3. Note that the $\vert$ and $+$ operators are no longer commutative and associative, and $\mathbf{0}$ is not the neutral element.

An Interpretation of the Proof Terms. Since the parallel operator has no congruence rules, we can interpret sequences of parallel tags as abstract addresses that uniquely identify sequential subprocesses. For example, consider the process

Table 4. Proved Transition system of the π-calculus.

$$Act : \mu.P \xrightarrow{\mu} P$$

$$Sum_0 : \frac{P \xrightarrow{\theta} P'}{P + Q \xrightarrow{+_0\theta} P'} \qquad Par_0 : \frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{\|_0\theta} P'|Q}, bn(\ell(\theta)) \cap fn(Q) = \emptyset$$

$$Sum_1 : \frac{Q \xrightarrow{\theta} Q'}{P + Q \xrightarrow{+_1\theta} Q'} \qquad Par_1 : \frac{Q \xrightarrow{\theta} Q'}{P|Q \xrightarrow{\|_1\theta} P|Q'}, bn(\ell(\theta)) \cap fn(Q) = \emptyset$$

$$Com_0 : \frac{P \xrightarrow{\overline{x}z} P', Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\langle\|_0\overline{x}z,\|_1 x(y)\rangle} P'|Q'\{z/y\}} \qquad Open : \frac{P \xrightarrow{\overline{x}\,y} P'}{(\nu y)P \xrightarrow{\overline{x}\langle y\rangle} P'}, y \neq x$$

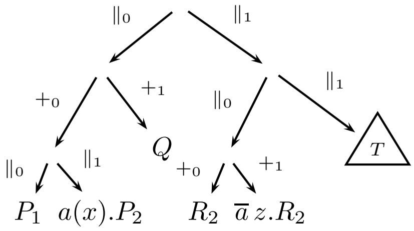
$$Com_1 : \frac{Q \xrightarrow{x(y)} Q', P \xrightarrow{\overline{x}z} P'}{Q|P \xrightarrow{\langle\|_0 x(y),\|_1\overline{x}z\rangle} Q'\{z/y\}|P'} \qquad Res : \frac{P \xrightarrow{\theta} P'}{(\nu x)P \xrightarrow{\theta} (\nu x)P'}, x \notin n(\ell(\theta))$$

$$Close_0 : \frac{P \xrightarrow{\overline{x}(z)} P', Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\langle\|_0\overline{x}(z),\|_1 x(y)\rangle} (\nu y)(P'|Q'\{z/y\})}$$

$$Close_1 : \frac{Q \xrightarrow{x(y)} Q', P \xrightarrow{\overline{x}(z)} P'}{Q|P \xrightarrow{\langle\|_0 x(y),\|_1\overline{x}(z)\rangle} (\nu y)(Q'\{z/y\}|P')}$$

$$S = ((P_1 \,|\, a(x).P_2) + Q) \,|\, ((R_1 + \overline{a}\,z.R_2) \,|\, T)$$

whose syntax tree is



The process $(P_1|a(x).P_2)+Q$ has $\|_0$ as abstract address, while the tree associated to the sub process T is identified by the abstract address $\|_1\|_1$.

Below we introduce the function ∂ for extracting abstract addresses from the proof terms; there is no need to define the function ∂ on pairs because it will be always applied component-wise.

Definition 5. *The function ∂ is inductively defined on proof terms:*

$$\begin{array}{rcl} \partial(\|_i\, \vartheta\mu) & = & \|_i\, \partial(\vartheta\mu) \\ \partial(\mu) & = & \epsilon \\ \partial(+_i\vartheta\mu) & = & \epsilon \end{array}$$

For example, consider the transition $S \xrightarrow{\langle \|_0 +_0 \|_1 a(x), \|_1 \|_0 +_1 \overline{a}\, z \rangle} (P_1 \mid P_2) \mid (R_2 \mid T)$ The proof terms of the communication contain the unique abstract addresses: $\partial(\|_0 +_0 \|_1) = \|_0\|_1$ for the input action and $\partial(\|_1\|_0 +_1) = \|_1\|_0$ for the output action.

5.1 Stochastic Semantics

As it happens for PEPA nets, we associate probabilistic information with actions. Hence, a random variable X_θ, which expresses the time duration of the action described by θ, must be associated to each proof term θ. The values that X_θ can assume are regulated by an exponential function $f_\theta(x) = \lambda e^{-\lambda x}$. Our approach mainly differs from the PEPA one because we do not insert probabilistic parameters in the syntax of the calculus, but we derive them from proof terms. The basic idea is that the operational semantics defines abstract machines and the proofs of transitions (encoded in proof terms) represent the low level routines of the abstract machine needed to implement transitions. We then assign rates to single tags (low level routines) and we give a way of composing them in order to compute rates of the transitions. Thus, for example, an action fired after a choice costs more than the same action occurring deterministically. Therefore, we have two logical phases. First we describe the system functionalities with a specification language. Then we associate quantitative values with the actions of the specification through an interpretation function of the proof terms. Such interpretation function is called *rate* function. Once rates have been associated with transitions, we derive CTMC and perform numerical analysis using the same techniques described in the previous section. Below we give the definition of the rate function that will be used in our case study.

A Rate Function. We assume that the throughput[1] of the communication channels and the size of the messages exchanged are given. We will use two auxiliary functions: th and $size$, for associating to each proof term a throughput and a size measure. The function th associates a throughput with the triple $(\vartheta_0, \vartheta_1, name)$, where ϑ_0 and ϑ_1 are the abstract addresses of the subprocesses that are communicating. The parameter $name$ represents the channel that the partners in a communication use to interact. The function $size$ associates a byte size with the couple $(\vartheta, name)$, where $name$ is the data sent and ϑ is the abstract

[1] The number of bits, characters, or blocks passing through a data communication channel. Throughput may vary greatly from its theoretical maximum. Throughput is expressed in data units per period of time; *e.g.* as blocks per second.

address of the sender process. We use also the function *min* which returns the minimum value between its two arguments.

The definition of the rate function is given for the asynchronous and for the synchronous case:

$$\$(\vartheta\mu) = \frac{size(obj(\mu))}{th(\partial(\vartheta), \epsilon, sbj(\mu))} \times \$_o(\vartheta)$$

$$\$(\vartheta\langle\vartheta_0\mu_0, \vartheta_1\mu_1\rangle) = \frac{size(obj(\mu_i))}{th(\partial(\vartheta\vartheta_0), \partial(\vartheta\vartheta_1), sbj(\mu_i))} \times min(\$_o(\vartheta\vartheta_0), \$_o(\vartheta\vartheta_1))$$

where the value returned by the auxiliary function $\$_o$ represents a *slowing factor* due to the time spent by the run time support. Consider the case when the proof terms record a communication as in $\vartheta\langle||_0\vartheta_0\mu_0, ||_1\vartheta_1\mu_1\rangle$ (for the other asynchronous case similar, yet simpler, considerations hold). The two partners perform independently some low-level operations locally to their environment. These operations are recorded in ϑ_0 and ϑ_1, inductively built by the application of the rules that fill in the premises of rules *Com* or *Close*. Each of the ϑ_i leads to a delay in the rate of the corresponding μ_i, which we compute through the auxiliary cost function $\$_o$. Then the pairing $\langle||_0\vartheta_0\mu_0, ||_1\vartheta_1\mu_1\rangle$ occurs and corresponds to the actual communication. Finally, there are those operations, recorded in ϑ, that account for the common context of the two partners. Also, the slow down due to this common context is computed using $\$_o$. Since communication is synchronous and handshaking, we take the minimum of the costs of the operations performed by the participants independently (originated by ϑ_i) to make communications reflect the speed of the slower partner[2].

For example, if a proof term models a service request from a client to a server we could interpret the $+_0$ and $+_1$ tags, contained in the proof term portion of the server ϑ_i, as a time degradation factor due to the waiting time spent in queuing for accessing the server. Also, we can differentiate the slowing factor of an operator taking care of the position where it has been executed by relying on the parallel tags, *i.e.* we can associate with $\|_0\|_1 +_0$ and $\|_0\|_1\|_0 +_0$ different values. For simplicity, here we assume $\$_o(\vartheta) = 1$. A definition of $\$_o$ can be found in [17].

5.2 The π-Calculus Model of a Web Service

We now model the Web service presented in Sect. 2, made of five components: *Client*, *WebService*, *SOAPmsg*, *Discover* and *Database*. In our scenario we consider the Universal Description Discovery Integration (UDDI) registry, modeled by the process *Discover*, which provides to the client the description of the web service. Moreover we assume that the *WebService* process queries a remote database, described as the process *Database*, in order to execute its task.

For the sake of readability, we write *x_a* for a place-holder that will be replaced with the value *a*.

---
[2] Recall that the lower the cost, the greater the time needed to complete an action and hence the slower the speed of the transition occurring.

The *Discover* process interacts with the *Client* process by accepting the request *ask_Des* on the public channel *dis*. The name *ask_Des* represents the description of the service that the *Client* needs. The *Discover* sends back to the *Client* the private name *des* along the channel *ask_Des*. The name *des* represents the description of the service that the *Client* asked for.

$$Discover(dis) \stackrel{def}{=} dis(x\_askDes).(\nu\ des)\overline{x\_askDes}\, des.Discover(dis).$$

The *Client* process interleaves the activity of looking for web-services with various other activities that we express with output actions on the channel *localComp*. When the *Client* needs a web service, it sends the request *askDes* to the *Discover* on the public channel *dis*. Then, the *Client* sends the description *des*, received by the discovery service, to the *SOAPmsg* on the public channel *client*. After that, it executes local operations, $\overline{localComp}$, until it receives the answer *service* from the *SOAPmsg* along the private channel *des*.

$$\begin{aligned} Client(localComp, dis, client, y) \stackrel{def}{=}\ & \overline{localComp}\, y.Client(localComp, dis, client, y) \\ & + \\ & ((\nu\ askDes)\overline{dis}\, askDes.askDes(x\_des). \\ & \overline{client}\, x\_des.Client_2(des, localComp, y) \\ &) \end{aligned}$$

$$\begin{aligned} Client_2(des, localComp, y) \stackrel{def}{=}\ & \overline{localComp}\, y.Client_2(des, localComp, y) \\ & + \\ & des(x\_service). \\ & Client(localComp, dis, client, y) \end{aligned}$$

The *SOAPmsg* process uses the public channel *client* to accept a description of a service, *des*, from the *Client* seeking for a web service. Then, the *SOAPmsg* is ready to send to the *WebService* the new name *newdes*, representing the client's request, on the public channel *web*. The *SOAPmsg* will receive the answer, *service*, from *WebService* along the private channel *newdes* and then it will send back the answer to the *Client* along the private channel *des*.

$$\begin{aligned} SOAPmsg(client, web) \stackrel{def}{=}\ & client(x\_des).(\nu\, newdes)\overline{web}\, newdes. \\ & newdes(x\_service)\overline{x\_des}\, x\_service. \\ & SOAPmsg(client, web) \end{aligned}$$

The *WebService* process interacts with the *SOAPmsg* receiving the request on the public channel *web*. Then, it connects with a remote database, using the public channel *data* and the private channel *newdata*, to retrieve the information it needs to complete its task. Finally, it replies the answer *service* to the *SOAPmsg* on the channel *newdes*.

$$\begin{aligned} WebService(web, data) \stackrel{def}{=}\ & web(x\_newdes).(\nu\, newdata)\overline{data}\, newdata. \\ & newdata(x\_service).\overline{x\_newdes}\, x\_service. \\ & WebService(web, data) \end{aligned}$$

The *Database* process simply receives a request from the *WebService* along the public channel name *data* and then gives back the answer using the private channel *newdata*.

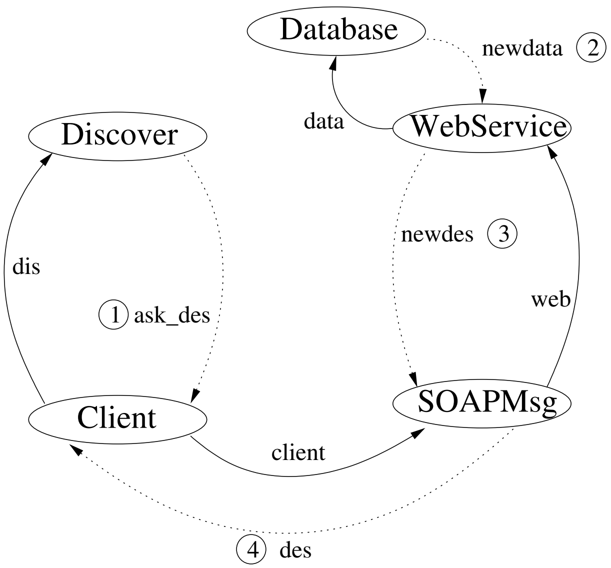


Fig. 4. The changes in the channel topology of the Web services example.

$$Database(data) \stackrel{def}{=} data(x\_newdata).(\nu\ service)\overline{x\_newdata}\, service.\ Database(data)$$

The complete system is given by composing in parallel the processes defined so far:

$$P(localComp, dis, client, y, web, data) \stackrel{def}{=} (Client(localComp, dis, client, y) \mid Discover(dis)) \mid (SOAPmsg(client, web) \mid (WebService(web, data) \mid Database(data)))$$

For the sake of readability from now onwards we shall write the process identifications omitting the parameters. Fig. 4 shows how the processes of the system interact. In particular the arcs between the processes are the communication channels: the ones with dotted lines are auxiliary channels and the circled numbers attached to the channels indicate the temporal order in which they are used.

The transition system generated by applying the enhanced operational semantics is illustrated in Fig. 5. Fig. 6 displays the processes passed through during a computation.

In order to obtain the rates for each proof term, we apply the rate function $ using the measures in Fig. 7. Recall that the function *th* associates a throughput to a pair of abstract addresses and a channel name; and that the function *size* associates the byte size of the data communicated to an abstract address and a data name, see Sect. 5.1. Some comments on the quantitative modeling of our example are in order. We do not need to associate a rate to the transitions corresponding to the execution of the asynchronous action $\overline{localComp}\, y$, because the

1	$-\ \Vert_0\Vert_0\ +_0\overline{localCompy} \rightarrow$	1
1	$-\ \Vert_0\ \langle \Vert_0\ +_1\overline{dis}\langle askDes\rangle, \Vert_1\ dis(x\_askDes)\rangle \rightarrow$	2
2	$-\ \Vert_0\ \langle \Vert_0\ askDes(x\_des), \Vert_1\ \overline{askDes}\langle des\rangle\rangle \rightarrow$	3
3	$-\langle \Vert_0\Vert_0\ \overline{client}\langle des\rangle, \Vert_1\Vert_0\ client(x\_des) \rightarrow$	4
4	$-\ \Vert_1\ \langle \Vert_0\ \overline{web}\langle newdes\rangle, \Vert_1\Vert_0\ web(x\_newdes)\rangle$	5
4	$-\ \Vert_0\Vert_0\ +_0\overline{localCompy} \rightarrow$	4
5	$-\ \Vert_1\Vert_1\ \langle \Vert_0\ \overline{data}\langle newdata\rangle, \Vert_1\ data(x\_newdata)\rangle \rightarrow$	6
5	$-\ \Vert_0\Vert_0\ +_0\overline{localCompy} \rightarrow$	5
6	$-\ \Vert_1\Vert_1\ \langle \Vert_1\ \overline{newdata}\langle service\rangle, \Vert_0\ newdata(x\_service)\rangle \rightarrow$	7
6	$-\ \Vert_0\Vert_0\ +_0\overline{localCompy} \rightarrow$	6
7	$-\ \Vert_1\ \langle \Vert_0\ newdes(x\_service), \Vert_1\Vert_0\ \overline{newdes}\langle service\rangle\rangle \rightarrow$	8
7	$-\ \Vert_0\Vert_0\ +_0\overline{localCompy} \rightarrow$	7
8	$-\langle \Vert_0\Vert_0\ des(x\_service), \Vert_1\Vert_0\ \overline{des}\langle service\rangle\rangle \rightarrow$	1

Fig. 5. Transition system of the process P.

self-loops are not represented by the Markov chain model. We assume that the communication connection from the *Client* to the *Discover* has the throughput of $300 Kbyte/sec$, and that the communication connection from the *Discover* to the *Client* has a slower throughput of $250 Kbyte/sec$ (first and second rows of the tables in Fig. 7). We assume that the communication between the *Client* and the *WebService* is asynchronous, thus we consider that the *Client* requires a shorter time to send the *SOAPmsg* to the *WebService* than the time the *WebService* needs to receive the *SOAPmsg* (third and fourth rows of the tables in Fig. 7). The difference between the sending time and the receiving time could be sensible if the *Client* and the *WebService* are located in two distant geographic regions or else if there are waiting queues for accessing the *WebService*. Recall that the *SOAPmsg* process represents a message which is sent by the *Client* to the *WebService* and then by the *WebService* to the *Client*. The communications between the *WebService* and the *Database* model the time spent by the *WebService* to execute its task (fifth and sixth rows of the tables in Fig. 7). For the last two communications between the *WebService* and the *SOAPmsg* and between the *SOAPmsg* and the *Client* we again suppose that the sending time of the *SOAPmsg* is shorter than the correspond receiving time.

In Fig. 7 we also associate quantitative measures to the private names *ask_Des*, *newdata*, *newdes* and *des*, (when used as data) assuming that they will not vary significantly from one execution of the system to another.

Fig. 8 shows the infinitesimal generator matrix $\mathbf{Q}$ of the CTMC. We solve $\pi\mathbf{Q} = \mathbf{0}$ subject to $\sum \pi = \mathbf{1}$, obtaining the vector
$\pi = (0.014, 0.033, 0.05, 0.063, 0.05, 0.031, 0.255, 0.212)$ as stationary distribution.

Adopting the technique for computing the rewards, described in [4], we can analyze for instance the probability of using the data name *des*. We consider the following reward array $(0, 1, 0, 0, 0, 0, 0)$ that has value 0 everywhere and value 1 in the position of a state with an outgoing transition whose label contains the

1	$(Client \mid Discover) \mid (SOAPmsg \mid (WebService \mid Database))$
2	$(\ askDes(x\_des).\overline{client}\, x\_des.Client_2(\_des) \mid (\nu\ des)\overline{askDes}\, des.Discover)$ $\mid$ $(SOAPmsg \mid (WebService \mid Database))$
3	$(\ \overline{client}\, des.Client_2(des) \mid Discover) \mid (SOAPmsg \mid (WebService \mid Database))$
4	$(\ Client_2(des) \mid Discover)$ $\mid$ $((\nu\, newdes)\overline{web}\, newdes.newdes(x\_service).\overline{des}\, x\_service.SOAPmsg$ $\mid$ $(WebService \mid Database))$
5	$(\ Client_2(des) \mid Discover)$ $\mid$ $(\overline{newdes}\, x\_service.des(x\_service).SOAPmsg$ $\mid$ $((\nu\, newdata)\overline{data}\, newdata.newdata(x\_service).\overline{newdata}\, sevice.WebService) \mid Database))$
6	$(\ Client_2(des) \mid Discover)$ $\mid$ $(newdes(x\_service).\overline{des}\, x\_service.SOAPmsg$ $\mid$ $((\nu\, newdata)newdata(x\_service).\overline{newdata}\, x\_service.WebService$ $\mid$ $(\nu\, service)\overline{newdata}\langle service\rangle.Database))$
7	$(\ Client_2(des) \mid Discover)$ $\mid$ $(newdes(x\_service).\overline{des}\, x\_service.SOAPmsg \mid ((\nu\, newdata)\overline{newdes}\, service\, WebService) \mid Database))$
8	$(\ Client_2(des) \mid Discover) \mid \overline{des}\, service.SOAPmsg \mid (WebService) \mid Database))$

Fig. 6. Reachable state space of the π-calculus Web service model P.

name des as data (see the transition system in Fig. 5). Thus we obtain that the probability of the system to use the name des as data is 0.033. We also can rely on more specific tags of the proof terms, considering for example the probability that the $SOAPmsg$ and the $WebService$ are interacting. The correspond reward array is $(0, 0, 0, 1, 0, 1, 0)$ that has value 0 everywhere and value 1 in the position of a state with an outgoing transition located at $\|_1\|_0, \|_1\|_1\|_0$. The resulting probability is 0.318.

6 Conclusions

If they were to be viewed purely formally as high-level description languages for specifying continuous-time Markov chains, then PEPA nets and the Stochastic π-calculus would be considered equally expressive. That is to say, for a given CTMC C, it is possible to construct a high-level model in either formalism such that the underlying CTMC derived from the model is isomporphic to C. This is a fundamental agreement in expressive power, but it is a rather weak one, similar to the agreement that all programming languages are Turing complete. In this paper and in related work [3] we have sought to understand the connections between these formalisms more thoroughly.

The modelling paradigms supported by PEPA nets and the Stochastic π-calculus EOS approach have a common root in using interleaving models of

$th : Kbytes/secs.$	
$(\|_0\|_0, \|_0\|_1, dis)$	300.00
$(\|_0\|_0, \|_0\|_1, \text{askDes})$	250.00
$(\|_0\|_0, \|_1\|_0, client)$	250.00
$(\|_0\|_0, \|_1\|_0, web)$	200.00
$(\|_1\|_1\|_0, \|_1\|_1\|_1, data)$	250.00
$(\|_1\|_1\|_0, \|_1\|_1\|_1, \text{newdata})$	200.00
$(\|_1\|_0, \|_1\|_0, \text{newdes})$	250.00
$(\|_0\|_0, \|_1\|_0, \text{des})$	300.00

$size : Kbytes$	
$(\|_0\|_0, \text{askDes})$	10
$(\|_0\|_1, \text{des})$	20
$(\|_0\|_0, \text{des})$	30
$(\|_1\|_0, \text{newdes})$	30
$(\|_1\|_1\|_0, \text{newdata})$	30
$(\|_1\|_1\|_1, \text{service})$	150
$(\|_1\|_1\|_0, \text{service})$	150
$(\|_1\|_0, \text{service})$	150

Fig. 7. The functions *th* and *size* applied to the proof terms of the transition system in Fig. 5.

$$\begin{array}{ccccccc} 1 & \xrightarrow{30} & 2 & \xrightarrow{12.5} & 3 & \xrightarrow{8.3} & 4 \\ 6.66 \uparrow & & & & & & \downarrow 8.33 \\ 8 & \xleftarrow{1.33} & 7 & \xleftarrow{1.66} & 6 & \xleftarrow{2} & 5 \end{array}$$

Fig. 8. The CTMC of the Web services example.

concurrent systems to first describe and then analyse the temporal behaviour of global and mobile code applications. However, there are many opportunities in such an enterprise to exercise creativity in the expression of concepts such as process mobility and performance metrics over models of mobile code systems. The differences between the PEPA nets approach and the EOS approach highlight points where different design choices were made.

Inside the behavioural description of a system the modeller needs to represent sequential execution and causal ordering of events. Over this aspect of the behavioural modelling there is close agreement between PEPA nets and EOS. However, process algebras also need to represent the concurrent composition of sequential behaviours and concepts such as synchronisation, parameterisation, naming and scoping. In stochastically timed process algebras particularly there are many ways to design and justify the synchronisation operator for processes [10, 1] and different design decisions are naturally taken in the PEPA nets and EOS approaches.

Adjacent to this, and perhaps of greater importance, is the use of the process algebra machinery in defining the meaning of terms in the language and legitimising their analysis. The differences between PEPA nets and the EOS approach are most pronounced here. The EOS approach encodes the rules which are used to produce the derivatives of a process as proof terms in their derivations. This information is implicit in a PEPA net one-step derivation although a proof of any derivation could be obtained by revisiting the operational semantics of the language or by using an EOS semantics for PEPA nets [7]. The proof terms play a central role in the performance analysis process for EOS. The evaluation cost function is defined over the proof terms of the language and hence built into the language at the same level as the operational semantics. The cost function and the operational semantics interoperate, with structural congruence rules for operators being disabled by their use in the definition of the evaluation cost function.

In constrast for PEPA nets, performance measures over a model are defined outside the operational semantics for the language, and this separation is highlighted by the use of a separate logical language, PML_ν for the expression of these measures. This separation means that the interpretation of the language constructs is unchanged across models and so tools supporting the language can perform optimisations such as quotienting by PEPA's bisimulation equivalence [8]. This operation is performed by rewriting the terms denoting process derivatives to amalgamate syntactically distinct terms which represent processes which no external observer could distinguish. This has the effect of reducing the state space of the system and therefore reducing the numerical computation effort which is needed to find the steady-state probability distribution for a given assignment of values to the symbolic rates of the model.

Despite these differences in methodology the present paper illustrates that the two modelling approaches can be used effectively in modelling real-world global computing applications and complement each other well in practical use. Both of the modelling methods used here are continuing to develop both in theory and in practical application. When, as in the present paper, we can compare modelling idioms in use we have the opportunity to see how to import analysis methods and techniques from one formalism to the other, to the benefit of both.

Acknowledgements

The authors 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. J. T. Bradley and N. Davies. Reliable performance modelling with approximate synchronisations. In *Proceedings of the 7th International Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 99–118, Prensas Universitarias de Zaragoza, September 1999.
2. L. Brodo, S. Gilmore, J. Hillston, and C. Priami. A stochastic π-calculus semantics for PEPA nets. In *Proceedings of the workshop on Process Algebras and Stochastically Timed Activities (PASTA)*, pages 1–17, Edinburgh, Scotland, June 2002.
3. L. Brodo, S. Gilmore, J. Hillston, and C. Priami. Mapping coloured stochastic Petri nets to stochastic process algebras. In P. Kemper, editor, *Proceedings of the ICALP Workshop on Stochastic Petri nets*, June 2003. To appear.
4. G. Clark and J. Hillston. Towards automatic derivation of performance measures from PEPA models. In *Proceedings of UKPEW*, 1996.
5. P. Degano and C. Priami. Non-interleaving semantics for mobile processes. *TCS: Theoretical Computer Science*, 216, 1999.

6. P. Degano and C. Priami. Enhanced operational semantics: a tool for describing and analyzing concurrent systems. *ACM Computing Surveys*, 33(2):135–176, 2001.
7. S. Gilmore and J. Hillston. An enhanced operational semantics for PEPA nets. DEGAS project internal document, May 2002.
8. S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, 27(5):449–464, May 2001.
9. S. Gilmore, J. Hillston, and M. Ribaudo. PEPA nets: A structured performance modelling formalism. In T. Field, P.G. Harrison, J. Bradley, and U. Harder, editors, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in Lecture Notes in Computer Science, pages 111–130, London, UK, April 2002. Springer-Verlag.
10. J. Hillston. The nature of synchronisation. In U. Herzog and M. Rettelbach, editors, *Proceedings of the Second International Workshop on Process Algebras and Performance Modelling*, pages 51–70, Erlangen, November 1994.
11. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
12. J. Hillston and L. Kloul. Formal techniques for performance analysis: blending SAN and PEPA. Submitted for publication, 2002.
13. J. Hillston and L. Kloul. From SAN to PEPA: A technology transfer. In *Proceedings of the workshop on Process Algebras and Stochastically Timed Activities (PASTA)*, pages 56–76, Edinburgh, Scotland, June 2002.
14. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
15. R. Milner. *Communicating and Mobile Systems: The π Calculus.* Cambridge University Press, Cambridge, England, 1999.
16. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, September 1992.
17. C. Nottegar, C. Priami, and P. Degano. Performance evaluation of mobile processes via abstract machines. *IEEE Transactions on Software Engineering*, 27(10):867–889, 2001.
18. 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.
19. C. Priami. Language-based performance prediction for distributed and mobile systems. *Information and Computation*, 175(2):119–145, 2002.