Semantic-Driven Performance Evaluation
Extended Abstract

Chiara Nottegar[1], Corrado Priami[1], and Pierpaolo Degano[2]

[1] Dipartimento Scientifico Tecnologico, Università di Verona
Ca' Vignal 2, Strada Le Grazie 1, I-37134, Verona, Italy
priami@sci.univr.it
[2] Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56100 Pisa, Italy
degano@di.unipi.it

Abstract. We propose a structural operational semantics for mobile and distributed agents. From it we derive a stochastic transition system labelled by actions and their costs. These costs reflect the (net) architecture on which agents run. We then map stochastic transition systems to Markov chains, and performance evaluation is carried out using standard tools. The results of our approach are shown to agree with the ones obtained via classical evaluation techniques on a case study involving mobile computation.

1 Introduction

Recently, stochastic process algebras [10,11,2,4,15,17,18] have been proposed as a mean to specify software and to derive general performance measures. Process algebra specifications are built by combining the basic actions, often called prefixes, that a system performs. The occurrence of a prefix is represented by a transition labelled by the associated action. The stochastic variants of process algebras associate probabilistic distributions with prefixes. The transitions are then labelled by a pair $\langle \mu, r \rangle$, where μ represents the action performed and completed in a time drawn from an exponential distribution with parameter r. We easily derive a continuous time Markov chain from a stochastic transition system. The stationary distribution of the chain, if any, is computed through standard numerical tools. Finally, performance analysis is carried out on the basis of stationary distributions. The main limitation of this solution is that the designer of a system must specify its intended behaviour already having in mind all the features of the architecture on which the specification will be implemented. Otherwise there is little hope to associate suitable distributions with prefixes. But the less details are needed at specification time, the better.

Our idea is to retain the advantages of stochastic process algebras without modifying the syntax of the classic process algebras and thus letting specifications be independent of architectural aspects of implementations. The association of probabilistic distributions with prefixes is then a matter of the compiler

J.-P. Finance (Ed.): FASE'99, LNCS 1577, pp. 204–219, 1999.

or the interpreter of the calculus. A compiler necessarily has all the relevant information about the target architecture, so we can re-use it.

The main contribution of the paper is the development of an algebraic representation of systems which is independent of their run-time support. The way in which rates of transitions are computed allows us to take run-time support into account. The neat separation of functional and quantitative information permits to consider different architectures for the same system, and to establish the most adequate in a semiautomatic way.

We are mainly interested in specifying and evaluating distributed applications, possibly involving code migration. To present our proposal in a pure setting, we adopt here the higher order π-calculus ($HO\pi$) [19]. It is a basic language for mobility in which processes are first class values. Since our technique can be applied to any language with an operational semantics, no real limitation arises from our choice. Our proposal relies on the idea that performance evaluation and other quantitative analysis should start already at the design level. Besides robustness and reliability of the design, these early measures may save efforts. Indeed, if the design meets all behavioural requirements but leads to inefficient implementations, the system must be re-designed. This calls for the integration of behavioural and quantitative analysis of systems in a single methodology.

Our starting point is an enhanced version of operational semantics that gives raise to a hierarchy of specifications of distributed systems increasingly nearer to implementations [8,16]. The definition of the operational semantics follows the *SOS* style [14], where the activities of a system are represented by transitions deduced according to a set of inference rules driven by the syntax. More precisely, we exploit proved transition systems, a parametric model used in [7,9] to uniformly describe different qualitative aspects of processes. Transitions are labelled by encodings of their proofs. Intuitively, the proof of a transition can be interpreted as the low level routines performed by the run-time support to execute the transition. Then, by inspecting these rich labels we derive the costs of transitions, reflecting the target architecture. In our *SOS* definition of the operational semantics of $HO\pi$, the cost function is parametric and can be instantiated to reflect different target architectures. In this way, the same definition can describe the behaviour of a system running on different targets.

Many real languages for concurrency, like Facile, PICT, CML, are built on top of a core process calculus like the π-calculus [13] we use here, which can be seen as an intermediate language. Therefore, the compiled code of high level programs can be annotated with information about the physical architecture, along the lines that we propose here.

We end the paper with a case study involving mobile computation. The results given by our approach are shown to agree with those obtained via classical evaluation techniques [1]. This is a very small step towards the applicability of our approach. It is certainly the case that further case studies need to be done in order to refine the technique.

2 Higher Order π-Calculus

In this section we briefly recall the higher order π-calculus ($HO\pi$) [19], a model of concurrent communicating processes providing the notion of *naming*. Names can represent processes, and thus communications may cause processes to migrate. We slightly enrich its syntax to better express performance analysis.

Definition 1.
Let $\mathcal{N}$ *be a countable infinite set of* names $a, b, \ldots, x, y, \ldots$ *and let* $\mathcal{S}$ *be a countable infinite set of* invisible actions $\tau_0, \tau_1, \ldots$ *with* $\mathcal{N} \cap \mathcal{S} = \emptyset$. *We also assume a set of* agent identifiers, *each with an arity, ranged over by* $A, A_1, \ldots$.
Let $\mathcal{P}$ *be a countable infinite set of* processes $P, Q, R, \ldots$ *and let* $\mathcal{V}$ *be a set of* process variables $X, Y, \ldots$. *Let* K *stand for a process or for a name and let* U *stand for a process variable or for a name. Thus, we have the following syntax*

$$P ::= \mathbf{0} \mid X \mid \pi.P \mid \sum_{j \in J} P_j \mid P|P \mid (\nu x)P \mid [x = y]P \mid A(U_1, \ldots, U_n)$$

where π *may be either* $x(U)$ *for* input, *or* $\overline{x}K$ *for* output (*where* x *is the* subject *and* U *and* K *are the* object) *or* τ_i *for* silent *moves. Also,* J *in the summation above is a finite index set. Hereafter, the trailing* $\mathbf{0}$ *will be omitted.*

The prefix π is the first atomic action that the process $\pi.P$ can perform. The input prefix $x(U)$ binds the occurrences of U in the prefixed process P. Intuitively, some name or process U is received along the link named x. The output prefix $\overline{x}K$ does not bind the name or process K which is sent along x. A silent prefix τ_i denotes an action which is invisible to an external observer of the system. We use a set of silent prefixes because different internal actions may have different durations. Summation denotes nondeterministic choice. The process $\sum_{j \in J} P_j$ behaves as one among the P_j. The operator $|$ describes parallel composition of processes. The components of $P_1|P_2$ may act independently; also, an output action of P_1 (resp. P_2) at any output port $\overline{x}$ may synchronize with an input action of P_2 (resp. P_1) at x to create a silent action of the communication. The operator (νx) acts as a static binder for the name x in the process P that it prefixes. In other words, x is a unique name in P which is different from all the external names. The agent $(\nu x)P$ behaves as P except that actions at ports $\overline{x}$ and x are prohibited. However communications along link x of components of P are not prohibited because the resulting action will be a τ. We sometimes write $(\nu x, y)P$ for $(\nu x)(\nu y)P$. Matching $[x = y]P$ is an `if-then` operator: process P is activated if $x = y$. Each agent identifier A has a unique defining equation of the form $A(\tilde{U}) = P$ (hereafter, $\tilde{U}$ denotes $U_1, \ldots, U_n$), where the U_i are all distinct and are the only free names in P.

The operational semantics for the $HO\pi$ is defined in the *SOS* style. The metavariable for the labels of transitions is μ (it is distinct from π, the metavariable for prefixes, though it coincides in three cases). We introduce the set $\mathcal{A}$ of visible actions ranged over by α (i.e. $x(U)$ for input, $\overline{x}K$ for free output,

and $\overline{x}(K)$ for bound output). The bound output is originated by the interplay of output prefix and restriction as shown in Tab. 1.

We define our *enhanced labels*, in the style of [6,3,7]. The label of a transition records the inference rules used during its deduction, besides the action itself. It is then possible to derive different semantic models for $HO\pi$ by extracting new kinds of labels from the enriched ones (in [9] the last two authors studied qualitative aspects of the calculus). We call *proof term* the encoding of the proof in an enhanced label. Finally, we introduce a function ℓ that takes an enhanced label to the corresponding standard action label.

Definition 2. *Let* $\mathcal{L} = \{\|_0, \|_1\}$ *with* $\chi \in \mathcal{L}^*$, $\mathcal{O} = \{\sum_{m,h}, =_m, (\nu x), (\tilde{U})\} \ni o$ *and let* $\vartheta \in (\mathcal{L} \cup \mathcal{O})^*$. *Then the set* Θ *of* enhanced labels (*with metavariable* θ) *is defined by the following syntax*

$$\theta ::= \vartheta\alpha \mid \vartheta\tau_i \mid \vartheta\langle \|_0 \vartheta_0 \alpha_0, \|_1 \vartheta_1 \alpha_1 \rangle$$

with $\alpha_0 = x(U)$ *iff* α_1 *is either* $\overline{x}K$ *or* $\overline{x}(K)$, *and vice versa.*
Function ℓ *is defined as* $\ell(\vartheta\alpha) = \alpha$, $\ell(\vartheta\tau_i) = \ell(\vartheta\langle \|_0 \vartheta_0 \alpha_0, \|_1 \vartheta_1 \alpha_1 \rangle) = \tau$.

A tag $\sum_{m,h}$ means that a nondeterministic choice has been resolved in favour of the h^{th} component among m summands. Rule Par_0 (Par_1) adds to the label a tag $\|_0$ ($\|_1$) to record that the left (right) component is moving. Restriction is represented in labels to record that a filter has been passed. We record the resolution of a matching through tag $=_m$, where m is the size of the data to be compared. The rules Com_0 and $Close_0$ have in their conclusion a pair instead of a τ to record the components which interacted (and the proof of the relevant transitions). Their symmetric version Com_1 and $Close_1$ are obvious and are omitted. Finally, the invocation of a definition enriches the label of the conclusion of rule *Ide* with the tag $(\tilde{U})$. We also record in the labels the actual parameters $\tilde{U}$ because their number and size affect the instantiation cost.

Our transition system for $HO\pi$ is in Tab. 1, where an auxiliary transition relation $\xrightarrow{\theta}_I$ is used. The set I contains names that can occur in a communicated process and that are extruded. Rules *Close* use I to include the receiving process as well in the scope of the extruded names. Rule *Open* updates the set of these names, that *Close* empties (note that in rules *Open* and *Close* it is $I \subseteq fn(K)$). The actual transitions are generated by rule $HO\pi$ that discards index I. The transitions in the conclusion of each rule stand for all their variants. Recall that a variant of $P \xrightarrow{\theta} Q$ is a transition which only differs in that P and Q have been replaced by processes that are α-equivalent (they only differ in the choice of bound names), and $\ell(\theta)$ has been α-converted, where a name bound in $\ell(\theta)$ includes Q in its scope [13].

Hereafter, we write a transition $P \xrightarrow{\theta} Q$ simply as θ, when unambiguous. We also write $Ts(P_i)$ to denote the set of the transitions enabled in P_i. As usual, we denote transition systems by quadruple $\langle \mathcal{P}, \Theta, \longrightarrow, P \rangle$, where $\mathcal{P}$ is the set of states (processes), Θ is the labelling alphabet, $\longrightarrow$ is the transition relation defined in Tab. 1, and P is the initial state.

$$Act: \ \pi.P \xrightarrow{\pi}_{\emptyset} P$$

$$Sum: \ \frac{P_h \xrightarrow{\theta}_I P'_h}{\sum_{j\in J} P_j \xrightarrow{\sum_{|J|,h}\theta}_I P'_h}, \ h\in J \qquad Par_0: \ \frac{P \xrightarrow{\theta}_I P'}{P|Q \xrightarrow{\|_0\theta}_I P'|Q}, \ (bn(l(\theta))\cup I)\cap fn(Q)=\emptyset$$

$$Res: \ \frac{P \xrightarrow{\theta}_I P'}{(\nu x)P \xrightarrow{(\nu x)\theta}_I (\nu x)P'}, \ x\notin n(l(\theta)) \qquad Par_1: \ \frac{P \xrightarrow{\theta}_I P'}{Q|P \xrightarrow{\|_1\theta}_I Q|P'}, \ (bn(l(\theta))\cup I)\cap fn(Q)=\emptyset$$

$$Match: \ \frac{P \xrightarrow{\theta}_I P'}{[x=x]\,P \xrightarrow{=_{size(x)}\theta}_I P'} \qquad Open: \ \frac{P \xrightarrow{\vartheta\overline{x}K}_{\emptyset} P'}{(\nu I)P \xrightarrow{\vartheta\overline{x}(K)}_I P'}, \ x\notin I\subseteq fn(K)$$

$$Close_0: \ \frac{P \xrightarrow{\vartheta\overline{x}(K)}_I P' \ , \ Q \xrightarrow{\vartheta' x(U)}_{\emptyset} Q'}{P|Q \xrightarrow{\langle\|_0\vartheta\overline{x}(K),\|_1\vartheta' x(U)\rangle}_{\emptyset} (\nu I)(P'|Q'\{K/U\})}, \ fn(K)\cap fn(Q)=\emptyset$$

$$Com_0: \ \frac{P \xrightarrow{\vartheta\overline{x}K}_{\emptyset} P' \ , \ Q \xrightarrow{\vartheta' x(U)}_{\emptyset} Q'}{P|Q \xrightarrow{\langle\|_0\vartheta\overline{x}K,\|_1\vartheta' x(U)\rangle}_{\emptyset} P'|Q'\{K/U\}}$$

$$Ide: \ \frac{P\{\tilde{K}/\tilde{U}\} \xrightarrow{\theta}_I P'}{A(\tilde{K}) \xrightarrow{(\tilde{K})\theta}_I P'}, \ A(\tilde{U})=P$$

$$HO\pi: \ \frac{P \xrightarrow{\theta}_I P'}{P \xrightarrow{\theta} P'}$$

Table 1. Proved transition system of the HOπ-calculus.

The standard interleaving semantics is obtained from the proved transition system by relabelling each transition through function ℓ in Def. 2.

3 Stochastic Semantics

We first discuss how to assign costs to individual transitions. Then we show how to extract a continuous time Markov chain (CTMC) from a proved transition system. Finally we describe how to evaluate performance measures starting from a CTMC.

3.1 Cost Function

We now show how the parameter r of the action $\mu = \ell(\theta)$ from a label θ is derived. The intended meaning of r is that the execution of the action μ is completed in a time drawn from an exponential distribution with parameter r, called the *rate* of the transition. Indeed, this is the interpretation in classical stochastic process algebras, where the designer of a system assign a *fixed* rate to *all* the occurrences of μ. But this is seldom the case in real situations because

the actual cost of μ depends on the basic operations that the run-time support of the target architecture performs for firing μ. Typically, the resolution of a choice imposes some operations on the target architecture such as checking the ready list or implementing fairness policies. Therefore, in practice an action fired after a choice costs more than the same action occurring deterministically. The other operations of our calculus reflect analogous routines of the run-time support and delay the execution of an action as well – communications deserve a special treatment, see below. Therefore, we first assign a rate to the transition μ on a dedicated architecture that has only to perform μ. In other words, μ occurs in the empty context. We then model the performance degradation due to the run-time support by introducing a scaling factor for r for any operation of the routine implementing the transition θ. In this way, the new semantics takes into account the target architecture on which a system runs. Also, we automatically derive the distributions of transitions by inspecting the syntactical contexts where the actions which originate them are plugged. In fact, the context in which an action μ occurs in the program represents the operations that the target machine performs for firing μ just because the structural operational semantics of a language specifies its abstract machine. Accordingly, a suitable linearization of the deduction of a transition represents the execution of the corresponding run-time support routines on the target machine. As mentioned above, a proof term θ represents the context in which an action occurs and the proof of the transition it labels. Following this intuition and the discussion above, we assign a cost to each inference rule of the operational semantics via a function denoted by \$. In other words, the occurrence of a transition receives a duration time computed according to its deduction.

As a matter of fact, there is no need to fix here function \$, and we let it be a parameter to the definition of our model. In this way, it is possible to estimate the performance of different architectures, each with its own cost function.

We propose below a possible definition of \$ that considers some features of a somehow idealized architecture like the number of processors of a net or the bandwidth of a channel. Although not very accurate, they permit to derive performance measures that agree with the ones in the literature [1,5]. Other case studies are needed to tune the function \$.

We first assign costs to actions in $\mathcal{N} \cup \mathcal{S}$. When applied to a visible action $a \in \mathcal{A}$, our cost function returns the parameter of the exponential distribution which describes the time needed to perform the very basic, low-level operations corresponding to a, independently of the context in which a occurs. Similarly for the invisible actions τ_i that are not communications. We use a function $\$_\mu : \mathcal{N} \cup \mathcal{S} \to I\!R^+$ defined as

$$\$_\mu(\tau_i) = \lambda_i$$
$$\$_\mu(\overline{x}K) = f_{out}(bw(x), size(K))$$
$$\$_\mu(\overline{x}(K)) = f_{bo}(bw(x), size(K))$$
$$\$_\mu(x(U)) = f_{in}(bw(x), size(U))$$

The real numbers λ_i represent the cost of executing the routine corresponding to the i^{th} internal action τ_i. The functions f_{out}, f_{bo} and f_{in} define the costs of the routines which implement the send and receive primitives. Function f_{bo} differs from f_{out} because y must be a fresh name, so it incorporates a call to a name generator gs (i.e., $f_{bo} = f_{out} + gs$). Besides the implementation cost due to the algorithms of send and receive, the functions above depends on the bandwidth of the communication channel ($bw(x)$) and the size of the objects transmitted.

According to the intuition that contexts slow down the speed of actions, we now determine a slowing factor (in $(0, 1]$) for any construct of the language. The cost of the operators in $\mathcal{L} \cup \mathcal{O}$ is expressed by the function $\$_o : \mathcal{L} \cup \mathcal{O} \to (0, 1]$

$$\$_o(\textstyle\sum_{m,j}) = f_+(m)$$
$$\$_o(||_i) = f_{||}(np), \quad i = 0, 1$$
$$\$_o(=_{size(x)}) = f_=(size(x))$$
$$\$_o((\nu x)) = f_\nu(n(P))$$
$$\$_o((U_1, \ldots, U_m)) = f_{()}(size(U_1), \ldots, size(U_m), np)$$

We have explicitly used a minimal set of parameters affecting the cost of the routines implementing any operator. In particular, the resolution of a choice implementing fairness policies is affected at least by the number of summands. Parallel composition is evaluated according to the number np of processors available. A particular case is $\$_o(||) = 1$ that arises when there is an unbound number of processors. (Recall that we are not yet considering communications.) The cost of matching depends on the size of the data to be compared. The cost of restriction depends at least on the number of names in a process because its resolution needs a search in a table of names. Finally, the activation of a new process via a constant invocation has a cost depending on the size and the number of the actual parameters as well as on the number of processors available.

We now consider synchronizations. To determine their impact on the overall cost function, we follow their deduction. Essentially, the two partners perform independently some low-level operations locally to their environment. These operations (call them *pre-synch*) correspond to the rules applied to fill in the premises of rules *Com* and *Close*. The application of either of them is recorded by pairing the proof terms corresponding to the pre-synch operations. Note that *Com* or *Close* rules can be applied exactly once in a derivation (see Tab. 1). Afterwards, some operations common to both partners are needed to derive the actual transition representing the communication. The cost of these additional operators is derived using $\$_o$.

We compute the slow down factor due to communication as follows. Since it is synchronous and handshaking, we first take the minimum of the costs of the pre-synch operations performed by the participants independently. Thus a communication reflects the speed of its slower partner.

We then use a function $f_{\langle\rangle} : \mathcal{L}^* \times \mathcal{L}^* \to (0, 1]$ to take the distance of the partners into account. For encoding locations, we use here $\chi_0, \chi_1 \in \{||_0, ||_1\}^*$. To see why, consider the binary abstract syntax tree of a process, when the parallel

composition | is the only syntactic operator. Then, a sequence χ can be seen as the access path from the root, i.e. from the whole process, to a leaf, i.e. to a sub-process. So, the two arguments of $f_{\langle\rangle}$, together with allocation tables, can be used to determine where the two communicating processes actually reside.

To apply function $f_{\langle\rangle}$, we need an auxiliary function $\underline{\cdot} : (\mathcal{L} \cup \mathcal{O})^* \to \mathcal{L}^*$ that extracts the parallel tags from proof terms, inductively defined as (ϵ is the empty string)

$$\underline{\epsilon} = \epsilon, \quad \underline{||_i \vartheta} = ||_i \underline{\vartheta}, \quad \underline{o\vartheta} = \underline{\vartheta}$$

We can now define the function that maps a proof term θ to a pair $\langle \ell(\theta), \$(\theta) \rangle$, by giving the function \$. It is defined inducing on θ and using the auxiliary functions $\$_\mu$ as basis and $\$_o$ and $f_{\langle\rangle}$. The function $\$: \Theta \to \mathbb{R}^+$ is defined as follows

$$\$(\mu) = \$_\mu(\mu)$$
$$\$(o\theta) = \$_o(o)\$(\theta)$$
$$\$(\langle \vartheta_0 a_0, \vartheta_1 a_1 \rangle) = f_{\langle\rangle}(\underline{\vartheta_0}, \underline{\vartheta_1}) \times \min\{\$(\vartheta_0 \alpha_0), \$(\vartheta_1 \alpha_1)\}$$

3.2 Interpretation of Costs

The interpretation of a transition $P_0 \xrightarrow{\theta} P_1$ is as follows. The action $\ell(\theta)$ has to wait a delay Δt drawn from the exponential distribution with parameter $\$(\theta)$ before its actual completion. In other words, Δt may be seen as the duration of the transition.

The dynamic behaviour of processes is determined by a *race condition*. All activities enabled attempt to proceed, but only the fastest one succeeds. The fastest activity is different on successive attempts because durations are expressed by random variables. The continuity of probabilistic distributions ensures that the probability of two activities ending simultaneously is zero. Furthermore, exponential distributions enjoy the *memoryless property*. Roughly speaking, the time at which a transition occurs is independent of the time at which the last transition occurred. Thus, the time elapsed by an activity in a state in which another one is the fastest is useless. This means that any time a transition becomes enabled, it restarts its elapsing time as it were the first time that it is enabled.

The race condition replaces nondeterministic choices with probabilistic ones. In fact, we can associate probabilities to transitions as follows. We define the *exit rate* of P (written $r(P)$) as the sum of the rates of all activities that are enabled in P, i.e. $r(P) = \sum_{\theta_i \in Ts(P)} \$(\theta_i)$. The occurrence probability of a transition $P \xrightarrow{\theta} P'$ is the ratio between its rate and the *exit rate* of P, i.e. $\$(\theta)/r(P)$. Consider the process $P = a + (a + b)$. If we define the cost function as

$$\$(a) = 1, \quad \$(b) = 3, \quad \$(+_i) = 1/2$$

the costs of the transitions are

$$\$(+_0 a) = 1/2, \quad \$(+_1 +_0 a) = 1/4 \quad \$(+_1 +_1 b) = 3/4.$$

Thus the exit rate of P is $3/2$, and the probability that P fires the transition $+_0 a$ is $1/3$. Following [11], we need the *apparent rate* of an action a in a given process P, $r_a(P)$. It is the sum of the rates of all actions a that are enabled in P, i.e. $r_a(P) = \sum_{\substack{\theta_i \in Ts(P) \\ l(\theta_i)=a}} \$(\theta_i)$. For instance, the apparent rate of a in P is $3/4$. This is the rate captured by an external observer of the system, that can only register actions and their occurrence frequency. Apparent rate allows us to compute *conditional probabilities*, as well. In fact, the probability of a transition $P \xrightarrow{\theta} P'$, with $l(\theta) = a$, given that an action a occurs, is $\$(\theta)/r_a(P)$. For example, the probability of the transition $+_0 a$ in P, given that an a occurs, is $2/3$. Therefore, the rate of a transition is its occurrence probability times its apparent rate. As usual, assume that parallel processes independently decide which actions to fire.

We recall the notion of stochastic process. A family of random variables $\{X(t)\ s.t.\ t \in T\}$ is a *stochastic process* with index set T. The set T is usually called *time parameter* and t is called *time*. The process is *continuous time* if T is a continuous set. The *state space* of the process is the set of possible values that $X(t)$ can assume. Intuitively, $X(t)$ is the state of the process at time t. Many systems arising in practice have the property that, once in a given state, the past states have no influence on the future. This is called the *memoryless* or *Markov property* and the stochastic processes satisfying it are called *Markov chains*, if their state space is discrete.

The following theorem suggests how to turn a process into a *CTMC*. We restrict ourselves to processes that originate a finite state space and that are closed (i.e they contain no free name). The proof of Theorem 1 is a straightforward adaptation of the one of a similar result for *PEPA* [11]. We first introduce some auxiliary notation.

Given a transition relation $\longrightarrow$, we define $\longrightarrow^*$ as its reflexive and transitive closure, and we let the label of $\longrightarrow^*$, if any, be the string obtained by concatenation of the labels of the sequence of transitions $\longrightarrow$ in the closure. Also, we say that a process P_i is a *derivative* of a process P, if there exists a computation with source P and target P_i (in symbols, $P \overset{s}{\longrightarrow}^* P_i$). Then, given a process P, we let hereafter $d(P) = \{P_i \mid P \overset{s}{\longrightarrow}^* P_i\}$ be the set of all derivatives P_i of P. We can now state the theorem.

Theorem 1. *Given a process P, the stochastic process $\{X(t), t \geq 0\}$, where $X(t_i) = P_j$ means that process P at time t_i behaves as process P_j, is a continuous time Markov chain homogeneous in time with state space $d(P)$.*

We now define the instantaneous transition rate of the generator matrix at the level of the proved transition system. Recall that the transitions enabled in a process cannot be disabled by flow of time, i.e. the *CTMC* associated with a process is homogeneous in time. Therefore, the instantaneous transition rate from P_i to P_j is the sum of the rates of the transitions enabled in P_i and leading to P_j. More formally, we have the following proposition.

Proposition 1. *Given* $\langle \mathcal{P}, \Theta, \longrightarrow, P \rangle$, *let* $P_i, P_j \in \mathcal{P}$, *and let* $n = |d(P)|$. *Then, the generator matrix of the corresponding* CTMC *is a square matrix* $\mathbf{Q}$ $n \times n$ *whose elements* $q_{i,j}$ *are defined as*

$$q_{i,j} = \sum_{P_i \xrightarrow{\theta_n} P_j \in Ts(P_i)} \$(\theta_n), \text{ for } i \neq j, \text{ and } q_{i,i} = -\sum_{j \neq i}^{n} q_{i,j}$$

Since the equation in the above proposition defines the instantaneous transition rate from P_i to P_j in terms of the transitions of P_i, and since our semantics is finite branching (our systems are finite state and closed), we can define in *SOS* style the *CTMC* associated with a system. More precisely, we define a stratified transition system whose transition relation $\longrightarrow_M$ is defined in terms of $\longrightarrow$. We let the *CTMC* of a process P_i (written $CTMC(P_i)$) be the minimal transition graph defined by rule

$$CTMC : \frac{P_i \xrightarrow{\theta} P_j}{P_i \xrightarrow{q_{i,j}}_M P_j}$$

where $q_{i,j}$ is defined according to Proposition 1.

3.3 Performance

We give a necessary condition for a process to originate a Markov chain with stationary distribution. Since the chains we consider have a finite state space, we are left to identify transition systems that guarantee the irreducibility of the corresponding Markov chains. If we call *cyclic* a state of a transition system that can be reached by any of its derivatives through a finite sequence of transitions, we have the following theorem.

Theorem 2. *Let* $\langle \mathcal{P}, \Theta, \longrightarrow, P \rangle$ *be a transition system with* P *cyclic. Then,* $CTMC(P)$ *is irreducible.*

Recall that if $CTMC(P)$ has a stationary distribution Π, it can be computed by solving the matrix equation

$$\Pi \mathbf{Q} = 0 \text{ with } \sum_i \Pi(P_i) = 1 \tag{1}$$

To get performance measures for process P, we associate a *reward* to each action a and we denote it as ρ_a, according to [12,11]. The reward of a process P is the sum of the rewards of the activities it enables, i.e. $\rho(P_i) = \sum_{\theta \in Ts(P_i)} \rho_{\ell(\theta)}$. The total reward of a component P is computed on the basis of an equilibrium distribution Π as

$$R(P) = \sum_{P \longrightarrow P_i} \rho(P_i) \Pi(P_i)$$

In the next section we report an example that shows how rewards are used to carry out performance evaluation.

4 A Case Study

In this section we use our mathematical framework to compare the relative efficiency of *remote procedure call* (*RPC*) and *remote evaluation* (*RE*) for a network management application presented in [1].

The management of networks based on *IP* uses the *simple network management protocol*. A *network management station* (*NMS*) calls some remote procedure for any server running on the network units which maintain a local data base with information for the maintenance of the network. The procedures above are typically *get* or *set* of values in the data base. This kind of interaction between *NMS* and the server is called *micro-management* and it generates a huge amount of network traffic.

Migration of code can reduce the traffic by sending to the server a piece of code that groups all the calls together. This approach is known as *management by delegation*. To compare our results with those of [1], we use their assumptions (reported below).

We assume that a data block of size X at the ISO/OSI application level generates at the network level an amount of data

$$X' = \alpha(X) + \beta(X)X$$

where $\alpha(X)$ is the cost of the *set-up* phase in a connection oriented protocol, and $\beta(X)$ is the size of the increment due to message encapsulation. For the sake of presentation, we adopt a single *overhead* function η such that

$$X' = \eta(X)X \quad \text{with } \eta(X) = \alpha(X)/X + \beta(X),\ \eta(X) > 1$$

and we write ηX for $\eta(X)X$.

We consider an application that collects data from N network units via Q remote procedure calls to each unit. We also assume that the interactions between *NMS* and any of the units is the same. Thus, we can safely specify the interactions with a single unit. The formal specification in $HO\pi$ is

$$NMS = \overline{a}\, i_1.\, a\,(R).\, \overline{a}\, i_2.\, a\,(R). \ldots . \overline{a}\, i_Q.\, a\,(R).\, NMS$$
$$D = a\,(I_k).\, \overline{a}\, r_k.\, D$$
$$Sys^{RPC} = (\nu a)\ (NMS | D)$$

NMS asks i_1 to D and waits for an answer. It repeats this task Q times. The unit D answers a datum r_k to a request i_k from the station. The channel a between *NMS* and D is made private via the operator (νa) in Sys^{RPC}. The corresponding proved transition system is depicted in Fig. 1 (a), where we omit $(\nu\, a)$ that prefixes all the labels.

Consider now the solution for code migration based on *remote evaluation* (*RE*). *NMS* sends a piece of code to D which queries *locally* the data base. A single message is needed to send back the Q answers to *NMS*. The specification in $HO\pi$ is

$$NMS = \overline{a}\, G.\, a\,(T).\, NMS \quad D = a\,(X).\, \overline{a}\, Qr.\, D \quad Sys^{RE} = (\nu a)\ (NMS | D)$$

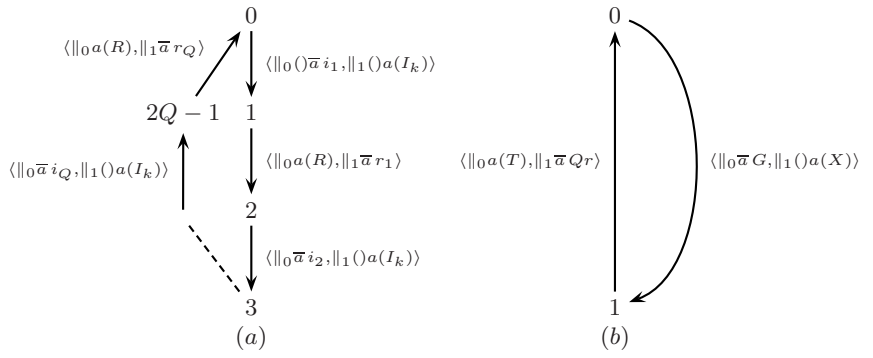

Fig. 1. Proved transition system of Sys^{RPC} (a) and of Sys^{RE} (b). We omitted $(\nu\, a)$ that prefixes all the labels.

and the corresponding proved transition system is in Fig. 1 (b). We assume that the agent is killed after the answers are sent back to *NMS*. This allows us to compare the *RPC* and *RE* solutions presented in [1].

According to our framework, we now define the cost function for the two solutions. We assume that requests and answers have average size I and R, respectively. Similarly, the average size of code migration is denoted by C. Let λ be the rate of sending one bit on the channel a; then the rate of sending an object on the same channel is the ratio of λ to the dimension (in bits) of the object. Thus, we have

$$\$(\overline{a}\, i_k) = \frac{\lambda}{\eta_{RPC} I},\ \$(\overline{a}\, r_k) = \frac{\lambda}{\tilde{\eta}_{RPC} R},\ \$(\overline{a}\, G) = \frac{\lambda}{\eta_{RE} C},\ \$(\overline{a}\, Qr) = \frac{\lambda}{\tilde{\eta}_{RE} QR}$$

where η_{RPC}, $\tilde{\eta}_{RPC}$, η_{RE} and $\tilde{\eta}_{RE}$ are the overhead of a single answer, a single request, a migration of code and of the Q answers sent back all together. The cost of input on a is related to that of output, but we have to take into account the substitutions of formal parameters with actual ones in the code of the process receiving the message. Thus we divide the cost of output by the number $k \geq 1$.

$$\$(a(x)) = \$(\overline{a}\, x)/k.$$

There is no rise of cost due to the parallel composition because there are two processors: one on the node of the *NMS* and one on that of D, so

$$\$_o(\|_i) = 1.$$

We indicate the rise of cost due to the restriction of the name a, to the invocation of identifiers and to the choice, respectively with

$$\$_o((\nu\, a)) = 1/n \text{ and } \$_o(()) = 1/p \text{ and } \$_o(+_i) = 1/l.$$

Now, supposing that $\|_0$ and $\|_1$ codify the names of the nodes of *NMS* and *D*, we assume

$$f_{\langle\rangle}(\|_0, \|_1) = 1/d$$

in order to take the distance between the two nodes into account. Finally, the costs associated to the transitions in the *RPC* case are

$$\$(\theta_{i,i+1}) = \frac{\lambda}{ndkp\eta_{RPC}I}, \ i = 0, 2, 4, .., 2Q-2$$

and

$$\$(\theta_{i,i+1}) = \frac{\lambda}{ndk\tilde{\eta}_{RPC}R}, \ i = 1, 3, 5, .., 2Q-1.$$

The same cost function \$ allows us to associate costs to the transitions of the *RE* specification. We have

$$\$(\theta_{0,1}) = \frac{\lambda}{ndkp\eta_{RE}C}, \quad \$(\theta_{1,0}) = \frac{\lambda}{ndk\tilde{\eta}_{RE}QR}.$$

We now can define the generator matrices $\mathbf{Q}^{RPC}$ and $\mathbf{Q}^{RE}$ of the two specifications according to Proposition 1. Since the initial states of both Sys^{RPC} and Sys^{RE} are cyclic, Theorem 2 ensures that the two systems have stationary distributions Π^{RPC} and Π^{RE}, and we can compute them according to equation (1). We yield,

$$\Pi^{RPC}(2i) = \frac{p\eta_{RPC}I}{Q(p\eta_{RPC}I + \tilde{\eta}_{RPC}R)} \quad \Pi^{RPC}(2i+1) = \frac{\tilde{\eta}_{RPC}R}{Q(p\eta_{RPC}I + \tilde{\eta}_{RPC}R)}$$

with $i = 0, \ldots, Q-1$, and

$$\Pi^{RE} = \left(\frac{p\eta_{RE}C}{p\eta_{RE}C + \tilde{\eta}_{RE}QR}, \frac{\tilde{\eta}_{RE}QR}{p\eta_{RE}C + \tilde{\eta}_{RE}QR} \right).$$

We end this section by comparing the efficiency of the two specifications. First we compute the throughput of both systems, i.e. the number of completed interactions *NMS-D* per unit time. Since in both systems each activity is visited only once, this throughput will be the same as the throughput of every activity. The throughput for an activity is found by associating a reward equal to the activity rate with each instance of the activity. So we compute the throughput of both systems by associating a reward equal to its rate to the first communication and a null reward to the other communications. Thus,

$$T^{RPC} = \sum_{i=0}^{2Q-1} \rho(i)\Pi^{RPC}(i) = \rho(0)\Pi^{RPC}(0) = \frac{\lambda}{ndkQ(p\eta_{RPC}I + \tilde{\eta}_{RPC}R)}$$

and

$$T^{RE} = \rho(0)\Pi^{RE}(0) + \rho(1)\Pi^{RE}(1) = \frac{\lambda}{ndk(p\eta_{RE}C + \tilde{\eta}_{RE}QR)}.$$

We can state that Sys^{RE} is better than or equivalent to Sys^{RPC} if $T^{RE} \geq T^{RPC}$. More precisely, we want

$$\frac{\lambda}{ndk(p\eta_{RE}C + \tilde{\eta}_{RE}QR)} \geq \frac{\lambda}{ndkQ(p\eta_{RPC}I + \tilde{\eta}_{RPC}R)}$$

hence

$$\eta_{RE}C \leq (Q\tilde{\eta}_{RPC}R - \tilde{\eta}_{RE}QR)/p + Q\eta_{RPC}l$$

Since the overhead η is fixed for any packet of a message, the longer the message, the lower the overhead is. Therefore, we usually have

$$Q\tilde{\eta}_{RPC}R \geq \tilde{\eta}_{RE}QR.$$

The analysis above suggests to use RE when the size of the agent sent is smaller than the sum of all the requests plus the difference of the overhead in the transmission of the answer. We finally note that by letting $p \simeq 1$ (because this cost is not considered in [1]), our results coincide with those in [1].

5 Conclusions and Further Work

The advantage of our approach is that it relies on a formal model which can be used as the kernel of a programming environment. In fact, the *CTMC*'s of the systems can be automatically derived from their transition systems. We only need to manipulate labels of transitions and to merge some arcs. Then, standard numerical packages can be used to derive performance measures. Note that the construction of the proved transition system only amounts to implement an interpreter for the semantic rules. Due to the syntax-directed way in which *SOS* semantics is defined, the interpreter can be easily written with a functional or logic language.

A calculus like the one we considered here can be the core of a real language (see Facile, PICT, CML, etc.). The compiled code can be annotated to make quantitative analysis. Also, it can be tranformed into syntactically different, yet behaviourally equivalent versions, via the usual equivalence laws (e.g. $P|(Q|R) = (P|Q)|R)$), before beginning to evaluate its performance. In this way, the compiler can help choosing not only the more suited architecture, but also the more adequate syntactic representation. This can be seen as a first step towards a code optimizer for concurrent languages, driven by semantics.

We tested our proposal on an application for network management presented in [1]. Under the same assumptions, our performance results agree with those of [1]. Lack of space prevents us from giving another example, taken from [5], for which again our method gives results in agreement with [5]. This makes us confident that our approach is applicable. Of course, we still need to apply our ideas to real size applications to test whether our method is scalable and to get more experience in the definition of reasonable cost functions.

Acknowledgments

The last author has been partially supported by the CNR Progetto Strategico *Modelli e Metodi per la Matematica e l'Ingegneria and by the MURST Progetto* Tecniche Formali per la Specifica, l'Analisi, la Verifica, la Sintesi e la Trasformazione di Sistemi Software.

References

1. M. Baldi and G.P. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In *Proceedings of ICSE'98*. ACM Press, 1998. 205, 209, 214, 215, 217
2. M. Bernardo, L. Donatiello, and R. Gorrieri. A formal approach to the integration of performance aspects in the modelling and analysis of concurrent systems. *Information and Computation*, 144:83–154, 1998. 204
3. G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, XI(4):433–452, 1988. 207
4. P. Buchholz. On a markovian process algebra. Technical report, Informatik IV, University of Dortmund, 1994. 204
5. A. Carzaniga, G.P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of ICSE'97*, pages 23–32. ACM Press, 1997. 209, 217
6. P. Degano, R. De Nicola, and U. Montanari. Partial ordering derivations for CCS. In *Proceedings of FCT, LNCS 199*, pages 520–533. Springer-Verlag, 1985. 207
7. P. Degano and C. Priami. Proved trees. In *Proceedings of ICALP'92, LNCS 623*, pages 629–640. Springer-Verlag, 1992. 205, 207
8. P. Degano and C. Priami. Enhanced operational semantics. *ACM Computing Surveys*, 28(2):352–354, 1996. 205
9. P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, march 1999. 205, 207
10. N. Götz, U. Herzog, and M. Rettelbach. TIPP- a language for timed processes and performance evaluation. Technical Report 4/92, IMMD VII, University of Erlangen-Nurnberg, 1992. 204
11. J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, Department of Computer Science, 1994. 204, 212, 213
12. R. Howard. *Dynamic Probabilistic Systems:Semi-Markov and Decision Systems*, volume II. Wiley, 1971. 213
13. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993. 205, 207
14. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Denmark, 1981. 205
15. C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995. 204
16. C. Priami. *Enhanced Operational Semantics for Concurrency*. PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1996. Available as Tech. Rep. TD-08/96. 205
17. C. Priami. Integrating behavioural and performance analysis with topology information. In *Proceedings of 29th Hawaian International Conference on System Sciences*, volume 1, pages 508–516, Maui, Hawaii, 1996. IEEE. 204

18. C. Priami. Enabling and general distributions in stochastic process algebras. In *Proceedings of Italian Conference on Theoretical Computer Science*, pages 192–203, Prato, 1998. World Scientific. 204
19. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992. 205, 206