Combining Timed Coordination Primitives and Probabilistic Tuple Spaces*

Mario Bravetti[1], Diego Latella[2], Michele Loreti[3],
Mieke Massink[2], and Gianluigi Zavattaro[1]

[1] Department of Computer Science, University of Bologna, Italy
{bravetti,zavattar}@cs.unibo.it
[2] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"- CNR
{Diego.Latella,massink}@isti.cnr.it
[3] Dipartimento di Sistemi e Informatica - Università di Firenze
loreti@dsi.unifi.it

Abstract. In this paper we present an integration of PLinda, a probabilistic extension of Linda, and StoKlaim, a stochastic extension of KLAIM. In the resulting language, StoPKlaim, the execution time of coordination primitives is modeled by means of exponentially distributed random variables, as in StoKlaim, the choice of the primitive to be executed among conflicting ones is thus resolved by the race condition principle, and the choice of the tuple to be retrieved by a single input/read operation in case of multiple matching tuples is governed by the weight-based probabilistic access policy of PLinda. The language represents a natural development and integration of previous results of the SENSORIA Project in the area of probabilistic and time-stochastic extensions of Tuple Space based coordination languages. The formal operational semantics of StoPKlaim is presented and an example of modeling is provided.

1 Introduction

Coordination languages play a central role in modeling and programming global-/service-oriented computers, their components and related applications. *Tuple Space* based coordination languages have proved very successful in this task [12,5,4]. In standard Tuple Space based coordination languages, when more than one tuple match the template of a data retrieval primitive, the tuple to be retrieved is chosen nondeterministically.

Performance, dependability, and in general quantitative, non-functional, aspects of system behaviour are of utmost importance for such systems, also in relation to their enormous size—networks typically consist of thousands or even millions of nodes—and their strong dependence on mobility and interaction. A specific task of the SENSORIA Project is to investigate the possibilities and implications of probabilistic and time-stochastic extensions of Tuple Space based

* Research partially funded by EU Integrated Project SENSORIA, contract n. 016004 and by Italian CNR/RSTL project XXL.

C. Kaklamanis and F. Nielson (Eds.): TGC 2009, LNCS 5474, pp. 52–68, 2009.

coordination languages. This has lead to the development of StoKlaim [11,6], a stochastic extension of KLAIM [5,2], and PLinda [3,4], a probabilistic extension of Linda [12]. In StoKlaim, every coordination primitive is assumed to have a random, exponentially distributed duration, the rate of which is explicitly specified in the primitive. In PLinda, weights are associated to tuples, so that each tuple matching a given template in input/read primitives will have a specific probability to be actually chosen.

In StoKlaim, if λ is the rate specified for a given data-retrieval primitive (e.g. a *in* or *read* primitive), the number of different tuples that match the template specified in the primitive influences the duration of the execution of the operation: if there are n different matching tuples, the speed of the data retrieval operation will have rate $\lambda \times n$. The different[1] tuples all have the same probability to be retrieved, i.e. $1/n$.

In PLinda, when a data retrieval operation is performed, the higher is the weight of one tuple, the greater is the probability for that tuple to be retrieved. More precisely, a tuple $\boldsymbol{d}$ with weight w is selected from the tuple space with probability $\frac{w}{W}$, where W is the total weight of all tuples matching the structure required by the retrieval operation.

The combination of the probabilistic and stochastic time features of PLinda and StoKlaim can be particularly useful in the context of global and service-oriented computing. For example, consider the problem of coordinating clients and services in a dynamic environment where new services can be created and old services may be disposed off. The service registry, where services are published by service providers and are retrieved by clients, can be profitably modeled by a tuple space. In order to guarantee a desired balance of workload, all tuples corresponding to different service providers for the same service can be enriched with different weights and, moreover, such weights can be modified dynamically, during system execution. Pragmatically, each provider may associate to the tuple which advertises its service a weight which quantifies (the inverse of) its current workload, as shown in e.g. [3]. Moreover, each coordination primitive can be tagged with a rate equal to the inverse of the (experimentally obtained) average duration of the execution of the primitive. This way interesting measures related to performance and dependability of the system can be computed and analysed on a model of the system, using traditional performance/dependability analysis techniques as well as advanced probabilistic and stochastic model-checking tools, as shown in e.g. [8].

In this paper we investigate the extension of StoKlaim with a refined probabilistic access to tuples inspired by that of PLinda. In this way, we obtain a more expressive modeling language in which the execution time of coordination primitives is modeled by means of exponentially distributed random variables, as in StoKlaim, the choice of the primitive to be executed among conflicting ones is thus resolved by the race condition principle, and the choice of the tuple to be retrieved by a single input/read operation in case of multiple matching

[1] Note that if there are several tuples with the same contents (hence, they are not "different" tuples), they are counted 1.

tuples is governed by the weight-based probabilistic access policy of PLinda. The language presented in this paper thus represents a natural development and integration of the previous results of the SENSORIA Project in the area of probabilistic and time-stochastic extensions of Tuple Space based coordination languages.

The proposed calculus has been also used for specifying and verifying a system where a set of distributed processes have to elect a leader (a uniquely designed process). The proposed specification is based on an adaptation of the asynchronous leader election protocol proposed in [17].

The operational semantics of StoPKlaim presented in this paper is defined using the technique proposed in [10] for a session-oriented calculus. In this approach, the transition relation associates each pair consisting of a StoPKlaim term and an action to a *function* from StoPKlaim terms to transition rates. This function maps each StoPKlaim term into the rate with which it can be reached from the term at the source of the transition, via the action. The technique handles in an elegant way the problem of transition multiplicity and two-party CCS-like interaction, which is typical of service oriented approaches as well as of Tuple Space based interaction primitives. It is compositional in nature and preserves commutativity and associativity of composition operators—in the context of two party synchronisation. This is of particular importance in the presence of process dynamic activation and mobility.

The rest of the paper is organized as follows. StoPKlaim is introduced in Section 2 while in Section 3 its stochastic semantics is defined. Section 4 shows how StoPKlaim can be used for modeling a simple case study while Section 5 concludes the paper.

2 StoPKlaim: Stochastic and Probabilistic Klaim

This section discusses the definition of StoPKlaim, the extension of StoKlaim, inspired by PLinda, in which tuples have an associated weight, and they are probabilistically retrieved in such a way that the higher is the weight, the greater is the probability for one tuple to be retrieved.

2.1 Syntax of StoPKlaim

Syntactic categories. We distinguish the following basic syntactic categories.

- $\mathcal{V}$, ranged over by $v, v', v_1, \ldots$, is a set of (basic data) *values*;
- $\mathcal{L}$, ranged over by $l, l', l_1, \ldots$, is a set of *logical addresses*, also called *localities*; the locality $\mathsf{self} \in \mathcal{L}$;
- $\mathcal{V}$-var, ranged over by $x, x', x_1, \ldots$, is a a set of *value variables*;
- $\mathcal{L}$-var, ranged over by $u, u', u_1, \ldots$, is set of *locality variables*;
- $\mathcal{P}$-var, ranged over by $X, X', X_1, \ldots$, be a set of *process variables*.

All the above sets are countable and are mutually disjoint. Let ℓ, ℓ', ℓ_1 range over $\mathcal{L} \cup \mathcal{L}$ -var. We will also use $e, e', e_1, \ldots$ to denote value expressions. The precise

syntax of expressions e is not specified since it is irrelevant for the purpose of the present paper. We assume that expressions contain, at least, basic values $\mathcal{V}$ and variables $\mathcal{V}$-var.

We adopt the $(\cdot)$-notation for sequences; e.g., $\boldsymbol{l} = l_1, l_2, \ldots, l_n$ denotes a sequence over $\mathcal{L}$ and $\boldsymbol{x} = x_1, x_2, \ldots, x_m$ is a sequence over $\mathcal{V}$-var. For sequence $\boldsymbol{s} = s_1, \ldots, s_n$, let $\{\boldsymbol{s}\}$ denote the set of elements in $\boldsymbol{s}$, i.e., $\{\boldsymbol{s}\} = \{s_1, \ldots, s_n\}$. One-element sequences and singleton sets are denoted as the element they contain, i.e., $\{s\}$ is denoted as s and $\boldsymbol{s} = s'$ as s'. The empty sequence is denoted by ϵ.

Nets and processes. Specifications in StoKlaim consist of nets and processes. The most elementary net is the null net, denoted $\mathbf{0}$. A net consisting of a single node with *locality* l is denoted $l :: E$ where E is a *node element*. In general, nets consist of several nodes composed in parallel. Thus, nets are constructed according to the following grammar:

$$\begin{array}{lll} N ::= & \mathbf{0} & \text{(null net)} \\ \mid & l :: E & \text{(node)} \\ \mid & N \parallel N & \text{(composition)} \end{array}$$

Node elements are either processes executing at a node—*process nodes* in the sequel—or data represented as a *weighted tuple* $\langle \boldsymbol{d} \rangle_w$ that is stored at a node, where d is a datum defined as follows:

$$d ::= P \mid l \mid v$$

and w is a weight representing the relative probability to retrieve the data tuple, with $w \in \mathbb{R}^+$, i.e. the set of strictly positive real numbers. We let $Nets$ denote the set of all StoPKlaim nets.

The definition of a node element is the following:

$$\begin{array}{lll} E ::= & P & \text{(running process)} \\ \mid & \langle \boldsymbol{d} \rangle_w & \text{(stored tuple)} \end{array}$$

Processes are built up from the terminated process **nil**, a set of randomly delayed actions, and standard process algebraic constructors such as prefix, choice, parallel composition and process instantiation with optional parameters. Formally, for action A:

$$\begin{array}{lll} P ::= & \mathbf{nil} & \text{(null process)} \\ \mid & (A, \lambda).P & \text{(action prefix)} \\ \mid & P + P & \text{(choice)} \\ \mid & P \mid P & \text{(parallel composition)} \\ \mid & \mathbf{rec} X.P & \text{(recursion)} \\ \mid & X & \text{(process variable)} \end{array}$$

The process $(A, \lambda).P$ executes action A with a random duration that is distributed exponentially with rate $\lambda \in \mathbb{R}^+$.

Templates. Tuples are retrieved from tuple spaces by using *T*emplates and Pattern Matching. A *Template field* can be a tuple field (a datum) or a binder. Template fields obey the following syntax:

$t ::= d$ (datum)
$\mid \quad !b$ (binder)

where $!b$ is a *binder*, i.e. a variable prefixed with an exclamation mark to indicate binding of such a variable defined by:

$!b ::= !X$ (process binder)
$\mid \quad !u$ (locality binder)
$\mid \quad !x$ (value binder)

Similarly, an unevaluated template field can be an unevaluated datum or a binder. Unevaluated template fields obey the following syntax:

$\underline{t} ::= \underline{d}$ (unevaluated datum)
$\mid \quad !b$ (binder)

where $\underline{d}$ is an unevaluated datum defined by

$\underline{d} ::= P$ (process)
$\mid \quad \ell$ (locality or locality variable)
$\mid \quad e$ (value expression)

Actions. A process can write tuple $\boldsymbol{d}$, with weight w, in repository l by the action $\mathbf{out}(\boldsymbol{d} : w)@l$. With an input action $\mathbf{in}(\boldsymbol{t})@l$ a process can withdraw a tuple that matches pattern, or *template* $\boldsymbol{t}$ from repository l. Processes can be written to/withdrawn from a repository as well.

Action $\mathbf{read}(\boldsymbol{t})@l$ is similar to $\mathbf{in}(\boldsymbol{t})@l$ except that the tuple at l is not deleted from the repository at l. The action $\mathbf{eval}(P)@l$ spawns process P at site l. A locality variable u can be used in place of l in all above actions. For the sake of simplicity, in this paper we do not consider the action for creating new nodes[2]. Actions are built according to the following grammar:

$A ::= \mathbf{out}(\underline{\boldsymbol{d}} : w)@\underline{l}$ (output)
$\mid \quad \mathbf{in}(\underline{\boldsymbol{t}})@\underline{l}$ (input)
$\mid \quad \mathbf{read}(\underline{\boldsymbol{t}})@\underline{l}$ (read)
$\mid \quad \mathbf{eval}(P)@\underline{l}$ (process spawning)

Well-formed specifications. Free and bound variables are defined in the usual way: in process $(\mathbf{in}(\boldsymbol{t})@\ell, r).P$ or $(\mathbf{read}(\boldsymbol{t})@\ell, r).P$ a binder occurring in $\boldsymbol{t}$ binds all free occurrences of the variable with the same name in P.

We say that a StoPKlaim *specification* N is *well-formed* if and only if it is type-correct and:

[2] Such actions do not involve tuples. Consequently, their semantics are not affected by the extension described in the present paper and their definition is as in [11,6].

- N does not contain free variables.
- In each recursion $\mathbf{rec}X.P$ process variable X occurs *guarded*, i.e. prefixed by an action, in P or as a tuple field of an **out** or **in** action, or as the argument of an **eval** action.
- In processes of the form $(\mathbf{in}(\boldsymbol{t})@\ell, r).P$ or $(\mathbf{read}(\boldsymbol{t})@\ell, r).P$, all binders occurring in $\boldsymbol{t}$ are distinct.
- Processes use only localities that really exist.

In the remainder of this paper we assume specifications to be *well-formed*.

Tuple evaluation. Function $[\![\cdot]\!]_.$ (cf. Table 1) evaluates tuples and templates. Notice that $[\![u]\!]$ yields u. In practice, the static semantics constraints together with the semantics of the **in** and **read** actions guarantee that variables are properly replaced by their values whenever necessary[3]. In Table 1 function $\mathcal{E}[\![\cdot]\!]$ is used for evaluating value expressions e. The definition of $\mathcal{E}[\![\cdot]\!]$ is outside the scope of the present paper.

Table 1. Tuple evaluation

$[\![P]\!]$	$\stackrel{\text{def}}{=} P$	$[\![!X]\!]$	$\stackrel{\text{def}}{=} !X$
$[\![\ell]\!]$	$\stackrel{\text{def}}{=} \ell$	$[\![!u]\!]$	$\stackrel{\text{def}}{=} !u$
$[\![e]\!]$	$\stackrel{\text{def}}{=} \mathcal{E}[\![e]\!]$	$[\![!x]\!]$	$\stackrel{\text{def}}{=} !x$
$[\![(t_1, \ldots, t_n)]\!]$	$\stackrel{\text{def}}{=} ([\![t_1]\!], \ldots, [\![t_n]\!])$		

Substitutions. In the inference rules defined below we exploit substitutions and combinations thereof. They have the usual meaning, i.e., for $d_1, \ldots, d_n$ ranging over $\mathcal{L} \cup \mathcal{V} \cup P$, and $w_1, \ldots, w_n$ ranging over $\mathcal{L}$-var $\cup$ $\mathcal{V}$-var $\cup$ $\mathcal{P}$-var, we let $[d_1/w_1 \ldots d_n/w_n]$, with $w_i \neq w_j$ for $i \neq j$, denote the substitution which replaces w_j by d_j for $0 < j \leq n$. Let $[]$ denote the empty substitution and, w.l.o.g, for substitution Θ_1:

$$[d_1/w_1, \ldots, d_n/w_n, d'_1/w'_1, \ldots, d'_m/w'_m]$$

and substitution Θ_2:

$$[d''_1/w'_1, .., d''_m/w'_m, d''_{m+1}/w'_{m+1}, .., d''_{m+h}/w'_{m+h}]$$

with $\{w'_{m+1}, \ldots, w'_{m+h}\} \cap \{ w_1, \ldots, w_n\} = \varnothing$, let $\Theta_1 \lhd \Theta_2$ be the substitution:

$$[d_1/w_1, \ldots, d_n/w_n, d''_1/w'_1, \ldots, d''_m/w'_m, d''_{m+1}/w'_{m+1}, \ldots, d''_{m+h}/w'_{m+h}] \quad .$$

[3] Thus, there is no need for explicitly evaluating variables by $[\![\cdot]\!]_.$.

Matching. Pattern-matching of templates with (stored) tuples is used to define the semantics of input and read-actions. In essence, this goes along similar lines as for KLAIM (and is rather standard). Function *match* (cf. Table 2) yields a substitution if a matching is successful.

Notice that for simplicity, only process binder variables match with processes. So, for instance, process constants, like e.g. **nil** do not match with themselves. This makes the definition of the matching function simpler.

Moreover, for the sake of notational simplicity, we overload the name *match* by using it also as a predicate: $match(\boldsymbol{t}, \boldsymbol{d})$ is true if and only if there exists Θ such that $match(\boldsymbol{t}, \boldsymbol{d}) = \Theta$.

Table 2. Pattern-matching of tuples against templates

$$match(l, l) \stackrel{\text{def}}{=} [] \qquad match(v, v) \stackrel{\text{def}}{=} []$$

$$match(!u, l) \stackrel{\text{def}}{=} [l/u] \qquad match(!x, v) \stackrel{\text{def}}{=} [v/x] \qquad match(!X, P) \stackrel{\text{def}}{=} [P/X]$$

$$\frac{match(t_1, d_1) = \Theta_1 \quad \ldots \quad match(t_n, d_n) = \Theta_n}{match((t_1, \ldots, t_n), (d_1, \ldots, d_n)) \stackrel{\text{def}}{=} \Theta_1 \lhd \ldots \lhd \Theta_n}$$

3 Stochastic Semantics of StoPKlaim

In this section we define the formal semantics of STOPKLAIM that permits associating a CTMC to each STOPKLAIM term. In order to illustrate the intended meaning, we first consider some simple example. Let us consider the following STOPKLAIM term

$$l :: (\mathbf{out}(v : w)@l, \lambda).\mathbf{nil}$$

Obviously, our interpretation of action rates requires that the associated CTMC contains two states, say s_0 and s_1, corresponding respectively to $l :: (\mathbf{out}(v : w)@l, \lambda).\mathbf{nil}$ and $l :: \langle v \rangle_w$, and one transition from s_0 to s_1, with rate λ. A similar CTMC would be associated to

$$l_1 :: (\mathbf{in}(!x)@l_2, \lambda).\mathbf{nil} | l_2 :: \langle v \rangle_w$$

Using the standard *race condition* principle for the Markovian interpretation of the non-deterministic choice composition, the CTMC associated to

$$l_1 :: (\mathbf{in}(!x)@l_2, \lambda_1).P_1 + (\mathbf{in}(!x)@l_2, \lambda_2).P_2 | l_2 :: \langle v \rangle_w$$

will have two transitions from the initial state, one labeled by λ_1 and the other one labeled by λ_2, thus with a total exit rate of $\lambda_1 + \lambda_2$. In particular, this implies that, in the degenerate case of $P_1 = P_2 = P$ and $\lambda_1 = \lambda_2 = \lambda$, the exit rate (as well as the rate labelling the *single* transition leaving the initial state) must be 2λ. A similar situation would arise with a term like

$$l_1 :: (\mathbf{in}(!x)@l_2, \lambda_1).P_1 | l_1 :: (\mathbf{in}(!x)@l_2, \lambda_2).P_2 | l_2 :: \langle v \rangle_w$$

in fact, in this term the race condition has an even stronger intuitive support: it comes from the competition of the two concurrent processes competing for downloading value v.

On the other hand, if one considers a term like

$$l_1 :: (\mathbf{in}(!x)@l_2, \lambda).P | l_2 :: \langle v_1 \rangle_{w_1} | l_2 :: \langle v_2 \rangle_{w_2}$$

the total exit rate will be λ. Indeed, in this case, only a single action with rate λ can be executed. However, two transitions can occur. One leading to $l_1 :: P[v_1/x] | l_2 :: \langle v_2 \rangle_{w_2}$ and the other leading to $l_1 :: P[v_2/x] | l_2 :: \langle v_1 \rangle_{w_1}$. Tuples $\langle v_1 \rangle_{w_1}$ and $\langle v_2 \rangle_{w_2}$ are selected probabilistically: the former with probability $\frac{w_1}{w_1+w_2}$, the latter with probability $\frac{w_2}{w_1+w_2}$. To correctly model the expected stochastic behaviour, rate λ is multiplied by the probabilities to select a given tuple. For these reasons, transitions above will occur with rates $\lambda \cdot \frac{w_1}{w_1+w_2}$ and $\lambda \cdot \frac{w_2}{w_1+w_2}$ respectively.

To guarantee correct modeling of race conditions, several approaches have been proposed in the past such as indexed or proved transitions [13,20], multirelations [16], recursive definitions of measures [21], multi-sets [14] and unique rate names [9]. In this paper stochastic behaviour of StoPKlaim is defined by using the approach proposed in [10]. In order to characterize the CTMC associated to each StoPKlaim system we first define a transition relation $\longrightarrow$ that associates to each process P and transition label γ a *next state function* (denoted by $\mathscr{P}$, $\mathscr{Q}$,...); such function associates each StoPKlaim net with an element of $\mathbb{R}_{\geq 0}$. Basically, $N \xrightarrow{\gamma} \mathscr{P}$ and $\mathscr{P}(N') = x \in \mathbb{R}_{>0}$ means that N' is reachable from N via the execution of γ. Value x can be either the duration of the associated action or the weight of the related tuple. On the other hand, $\mathscr{P}(N') = 0$ means that N' is not reachable from N via γ. This approach permits also *counting* the available tuples matching a given template, and *computing* the relative probabilities that one of these is selected.

We let Λ be the set of labels of the transitions; such labels are constructed according to

$$\begin{aligned}\gamma ::= {} & l_1 : \mathbf{o}(\boldsymbol{d})@l_2 \mid l_1 : \mathbf{e}(P)@l_2 \mid l_1 : \mathbf{i}(\boldsymbol{t} : \boldsymbol{d})@l_2 \mid l_1 : \mathbf{r}(\boldsymbol{t} : \boldsymbol{d})@l_2 \\ & \mid l_1 \rhd^{\boldsymbol{t}} \boldsymbol{d}@l_2 \mid l_1 \blacktriangleright^{\boldsymbol{t}} \boldsymbol{d}@l_2 \mid l_1 \lhd \boldsymbol{t}@l_2 \mid l_1 \blacktriangleleft \boldsymbol{t}@l_2\end{aligned}$$

where $l_1 : \mathbf{o}(\boldsymbol{d})@l_2$ and $l_1 : \mathbf{e}(P)@l_2$ identify output and eval actions; $l_1 : \mathbf{i}(\boldsymbol{t} : \boldsymbol{d})@l_2$ and $l_1 : \mathbf{r}(\boldsymbol{t} : \boldsymbol{d})@l_2$ identify input and read actions performed on template $\boldsymbol{t}$ and retrieving tuple $\boldsymbol{d}$. These labels model internal interactions. Label $l_1 \rhd^{\boldsymbol{t}} \boldsymbol{d}@l_2$ and $l_1 \blacktriangleright^{\boldsymbol{t}} \boldsymbol{d}@l_2$ model the offer of an input or read action performed by a process. These actions synchronize with $l_1 \lhd \boldsymbol{t}@l_2$ and $l_1 \blacktriangleleft \boldsymbol{t}@l_2$ that model the availability of tuples matching template $\boldsymbol{t}$.

Let us consider term $l_1 :: (\mathbf{out}(\boldsymbol{d} : w)@l_2, \lambda).P \parallel l_2 :: \mathbf{nil}$. As we suggested previously in this section, this term evolves, with rate λ, to $l_1 :: P \parallel l_2 :: \langle \boldsymbol{d} \rangle_w$. If we let $[N \mapsto \lambda]$ denote the function below:

$$[N \mapsto \lambda](N') = \begin{cases} \lambda & \text{if } N' = N \\ 0 & \text{otherwise} \end{cases}$$

the transition of interest is

$$l_1 :: (\mathbf{out}(\boldsymbol{d} : w)@l_2, \lambda).P \xrightarrow{l_1:\mathbf{o}(\boldsymbol{d})@l_2} [l_1 :: P \parallel l_2 :: \langle \boldsymbol{d} \rangle_w \mapsto \lambda]$$

while for all $\gamma \neq l_1 : \mathbf{o}(\boldsymbol{d})@l_2$ we have

$$l_1 :: (\mathbf{out}(\boldsymbol{d} : w)@l_2, \lambda).P \xrightarrow{\gamma} \varnothing$$

In what follows, we use the operations defined on next state functions given in Definition 1.

Definition 1. *Let $\mathscr{P}, \mathscr{Q} : Nets \to \mathbb{R}^{\top}_{\geq 0}$, $N, N_0, \ldots, M_n$ in $Nets$, and $\lambda_0, \ldots, \lambda_n$ in $\mathbb{R}^{\top}_{\geq 0}$; we let:*

- *$\varnothing$ denote the constant 0, so for each $N \in Nets$: $\varnothing(N) = 0$*
- *$[N_0 \mapsto \lambda_0, \ldots, N_n \mapsto \lambda_n]$ denotes function $\mathscr{P} : Nets \to \mathbb{R}^{\top}_{\geq 0}$ such that:*

$$\mathscr{P}(N) = \begin{cases} \lambda_i & \text{if } N = N_i \\ 0 & \text{otherwise} \end{cases}$$

- *$\mathscr{P} + \mathscr{Q}$ be the function $\mathscr{R} : Nets \to \mathbb{R}^{\top}_{\geq 0}$ such that:*

$$\mathscr{R}(N) = \mathscr{P}(N) + \mathscr{Q}(N)$$

- *$\mathscr{P} \parallel N_1$ be the function $\mathscr{R} : Nets \to \mathbb{R}^{\top}_{\geq 0}$ such that:*

$$\mathscr{R}(N) = \begin{cases} \mathscr{P}(N_2) & \text{if } N = N_2 | N_1 \\ 0 & \text{otherwise} \end{cases}$$

- *$\mathscr{P} \parallel \mathscr{Q}$ be the function $\mathscr{R} : Nets \to \mathbb{R}^{\top}_{\geq 0}$ such that:*

$$\mathscr{R}(N) = \begin{cases} \mathscr{P}(N_1) \times \mathscr{Q}(N_2) & \text{if } N = N_1 \parallel N_2 \\ 0 & \text{otherwise} \end{cases}$$

- *$\frac{\mathscr{P} \cdot w_1}{w_2}$ be the function $\mathscr{R} : Nets \to \mathbb{R}^{\top}_{\geq 0}$ such that:*

$$\mathscr{R}(N) = \begin{cases} \frac{\mathscr{P}(N) \cdot w_1}{w_2} & \text{if } w_2 \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- *$\oplus(\mathscr{P}) = \sum_{N \in Nets} \mathscr{P}(N)$*

Transition $\longrightarrow$ is formally defined by the rules in Table 3 and Table 4. These rules permit deriving with a single proof *all* the configurations which are reachable from a net with a given transition label.

Rules (Out) and (Eval) describe the behaviour of **out** and **eval**. The former states that $l_1 :: (\mathbf{out}(\boldsymbol{d} : w)@l, \lambda).P$ evolves to $l_1 :: P \parallel l_2 :: \langle \boldsymbol{d} \rangle_w$ with rate λ, while the latter states that $l_1 :: (\mathbf{eval}(Q)@l, \lambda).P$ evolves to $l_1 :: P \parallel l_2 :: Q$ with rate λ.

Table 3. StoPKlaim Operational Semantics: Positive Rules

$$l_1 :: (\mathbf{out}(\boldsymbol{d} : w)@l_2, \lambda).P \xrightarrow{l_1:\mathbf{o}(\boldsymbol{d})@l_2} [l_1 :: P \parallel l_2 :: \langle \boldsymbol{d} \rangle_w \mapsto \lambda] \quad (\text{Out})$$

$$l_1 :: (\mathbf{eval}(Q)@l_2, \lambda).P \xrightarrow{l_1:\mathbf{e}(Q)@l_2} [l_1 :: P \parallel l_2 :: Q \mapsto \lambda] \quad (\text{Eval})$$

$$\frac{\Theta = match(\boldsymbol{t}, \boldsymbol{d})}{l_1 :: (\mathbf{in}(\boldsymbol{t})@l_2, \lambda).P \xrightarrow{l_1 \rhd^{\boldsymbol{t}} \boldsymbol{d}@l_2} [l_1 :: P\Theta \mapsto \lambda]} \quad (\text{P-In})$$

$$\frac{\Theta = match(\boldsymbol{t}, \boldsymbol{d})}{l_1 :: (\mathbf{read}(\boldsymbol{t})@l_2, \lambda).P \xrightarrow{l_1 \blacktriangleright^{\boldsymbol{t}} \boldsymbol{d}@l_2} [l_1 :: P\Theta \mapsto \lambda]} \quad (\text{P-Read})$$

$$\frac{match(\boldsymbol{t}, \boldsymbol{d})}{l_2 :: \langle \boldsymbol{d} \rangle_w \xrightarrow{l_1 \lhd \boldsymbol{t}@l_2} [\mathbf{0} \mapsto w]} \quad (\text{T-In}) \qquad \frac{match(\boldsymbol{t}, \boldsymbol{d})}{l_2 :: \langle \boldsymbol{d} \rangle_w \xrightarrow{l_1 \blacktriangleleft \boldsymbol{t}@l_2} [l_2 :: \langle t \rangle_w \mapsto w]} \quad (\text{T-Read})$$

$$\frac{l :: P \xrightarrow{\gamma} \mathscr{P} \quad l :: Q \xrightarrow{\gamma} \mathscr{Q}}{l :: P + Q \xrightarrow{\gamma} \mathscr{P} + \mathscr{Q}} \quad (\text{Plus}) \qquad \frac{l :: P \xrightarrow{\gamma} \mathscr{P} \quad l :: Q \xrightarrow{\gamma} \mathscr{Q}}{l :: P|Q \xrightarrow{\gamma} \mathscr{P} \parallel l :: Q + l :: P \parallel \mathscr{Q}} \quad (\text{P-Par})$$

$$\frac{N_1 \xrightarrow{\gamma} \mathscr{P} \quad N_2 \xrightarrow{\gamma} \mathscr{Q} \quad \gamma \notin \{l_1 : \mathbf{i}(\boldsymbol{t} : \boldsymbol{d})@l_2, l_1 : \mathbf{r}(\boldsymbol{t} : \boldsymbol{d})@l_2\}}{N_1 \parallel N_2 \xrightarrow{\gamma} \mathscr{P} \parallel N_2 + N_1 \parallel \mathscr{Q}} \quad (\text{N-Par})$$

$$\frac{\begin{array}{c} N_1 \xrightarrow{l_1:\mathbf{i}(\boldsymbol{t}:\boldsymbol{d})@l_2} \mathscr{P}^s \;\; N_1 \xrightarrow{l_1 \rhd^{\boldsymbol{t}} \boldsymbol{d}@l_2} \mathscr{P}^i \;\; N_1 \xrightarrow{l_1 \lhd \boldsymbol{t}@l_2} \mathscr{P}^T \;\; N_1 \xrightarrow{l_1 \lhd \boldsymbol{d}@l_2} \mathscr{P}^o \\ N_2 \xrightarrow{l_1:\mathbf{i}(\boldsymbol{t}:\boldsymbol{d})@l_2} \mathscr{Q}^s \;\; N_2 \xrightarrow{l_1 \rhd^{\boldsymbol{t}} \boldsymbol{d}@l_2} \mathscr{Q}^i \;\; N_2 \xrightarrow{l_1 \lhd \boldsymbol{t}@l_2} \mathscr{Q}^T \;\; N_2 \xrightarrow{l_1 \lhd \boldsymbol{d}@l_2} \mathscr{Q}^o \end{array}}{N_1 \parallel N_2 \xrightarrow{l_1:\mathbf{i}(\boldsymbol{t}:\boldsymbol{d})@l_2} \frac{\mathscr{P}^s \parallel N_2 \cdot \oplus \mathscr{P}^T + N_1 \parallel \mathscr{Q}^s \cdot \oplus \mathscr{Q}^T + \mathscr{P}^i \parallel \mathscr{Q}^o + \mathscr{P}^o \parallel \mathscr{Q}^i}{\oplus(\mathscr{P}^T + \mathscr{Q}^T)}} \quad (\text{In})$$

$$\frac{\begin{array}{c} N_1 \xrightarrow{l_1:\mathbf{r}(\boldsymbol{t}:\boldsymbol{d})@l_2} \mathscr{P}^s \;\; N_1 \xrightarrow{l_1 \blacktriangleright^{\boldsymbol{t}} \boldsymbol{d}@l_2} \mathscr{P}^i \;\; N_1 \xrightarrow{l_1 \blacktriangleleft \boldsymbol{t}@l_2} \mathscr{P}^T \;\; N_1 \xrightarrow{l_1 \blacktriangleleft \boldsymbol{d}@l_2} \mathscr{P}^o \\ N_2 \xrightarrow{l_1:\mathbf{r}(\boldsymbol{t}:\boldsymbol{d})@l_2} \mathscr{Q}^s \;\; N_2 \xrightarrow{l_1 \blacktriangleright^{\boldsymbol{t}} \boldsymbol{d}@l_2} \mathscr{Q}^i \;\; N_2 \xrightarrow{l_1 \blacktriangleleft \boldsymbol{t}@l_2} \mathscr{Q}^T \;\; N_2 \xrightarrow{l_1 \blacktriangleleft \boldsymbol{d}@l_2} \mathscr{Q}^o \end{array}}{N_1 \parallel N_2 \xrightarrow{l_1:\mathbf{r}(\boldsymbol{t}:\boldsymbol{d})@l_2} \frac{\mathscr{P}^s \parallel N_2 \cdot \oplus \mathscr{P}^T + N_2 \parallel \mathscr{Q}^s \cdot \oplus \mathscr{Q}^T + \mathscr{P}^i \mid \mathscr{Q}^o + \mathscr{P}^o \mid \mathscr{Q}^i}{\oplus(\mathscr{P}^T + \mathscr{Q}^T)}} \quad (\text{Read})$$

$$\frac{l :: P[\mathbf{rec}X.P/X] \xrightarrow{\gamma} \mathscr{P}}{l :: \mathbf{rec}X.P \xrightarrow{\gamma} \mathscr{P}} \quad (\text{Rec})$$

Table 4. StoPKlaim Operational Semantics: $\varnothing$ Rules

$$\mathbf{0} \xrightarrow{\gamma} \varnothing \quad (\text{F-}\mathbf{0}) \qquad l_1 :: \mathbf{nil} \xrightarrow{\gamma} \varnothing \quad (\text{F-Nil})$$

$$\frac{\gamma \neq l_1 : \mathbf{o}(\boldsymbol{d})@l_2}{l_1 :: (\mathbf{out}(\boldsymbol{d} : w)@l, \lambda).P \xrightarrow{\gamma} \varnothing} \quad (\text{F-Out}) \qquad \frac{\gamma \neq l_1 : \mathbf{e}(Q)@l_2}{l_1 :: (\mathbf{eval}(Q)@l, \lambda).P \xrightarrow{\gamma} \varnothing} \quad (\text{F-Eval})$$

$$\frac{\gamma \neq l_1 \rhd^{\boldsymbol{t}} \boldsymbol{d}@l_1 \vee \neg match(\boldsymbol{t}, \boldsymbol{d})}{l_1 :: (\mathbf{in}(\boldsymbol{t})@l_2, \lambda).P \xrightarrow{\gamma} \varnothing} \quad (\text{F-In}) \qquad \frac{\gamma \neq l_1 \blacktriangleright^{\boldsymbol{t}} \boldsymbol{d}@l_1 \vee \neg match(\boldsymbol{t}, \boldsymbol{d})}{l_1 :: (\mathbf{read}(\boldsymbol{t})@l_2, \lambda).P \xrightarrow{\gamma} \varnothing} \quad (\text{F-Read})$$

$$\frac{\gamma \neq l_1 \lhd \boldsymbol{t}@l_2, l_1 \blacktriangleleft \boldsymbol{t}@l_2 \vee \neg match(\boldsymbol{t}, \boldsymbol{d})}{l_2 :: \langle \boldsymbol{d} \rangle_w \xrightarrow{\gamma} \varnothing} \quad (\text{F-Tuple})$$

Rule (P-In) models the behaviour of a process that performs an **in** action: if $\Theta = match(\boldsymbol{t}, \boldsymbol{d})$ then $(\mathbf{in}(\boldsymbol{t})@l_2, \lambda).P$ evolves with rate λ to $P\Theta$. Rule (P-Read) is similar. Notice that (P-In) and (P-Read) model the *intention* of a process to perform an **in** or a **read**. Availability of tuples at a given tuple space is modeled by means of rules (T-In) and (T-Read) that characterizes the tuples that permit executing an **in** and a **read** action respectively.

Rule (Plus) states that $l :: P + Q$ can behave either like $l :: P$ or like $l :: Q$. Moreover, rates of each transition are added to take multiplicity into account. Notice that function $\mathscr{P} + \mathscr{Q}$ ensures the race condition principle in a natural way. In particular, when both P and Q can perform the same action γ, $P + Q$ will perform that action with a rate which is the *sum* of the rates for γ in P and Q; this is a simple and elegant way for accounting for transition multiplicity. For instance:

$$\frac{\begin{array}{c} l_1 :: (\mathbf{eval}(Q)@l_2, .75).\mathbf{nil} \xrightarrow{l_1:\mathbf{e}(Q)@l_2} [l_2 :: Q \mapsto .75] \\ l_1 :: (\mathbf{eval}(Q)@l_2, 1.5).\mathbf{nil} \xrightarrow{l_1:\mathbf{e}(Q)@l_2} [l_2 :: Q \mapsto 1.5] \end{array}}{l_1 :: (\mathbf{eval}(Q)@l_2, .75).\mathbf{nil} + (\mathbf{eval}(Q)@l_2, 1.5).\mathbf{nil} \xrightarrow{l_1:\mathbf{e}(Q)@l_2} [l_2 :: Q \mapsto 2.25]} \text{(Sum)}$$

Rules (N-Par) and (P-Par) state that if γ is not a synchronisation action ($l_1 : \mathbf{i}(\boldsymbol{t} : \boldsymbol{d})@l_2$ or $l_1 : \mathbf{r}(\boldsymbol{t} : \boldsymbol{d})@l_2$), reachable states from $N_1 \parallel N_2$ (resp. $l :: P|Q$) by γ are those reachable from N_1 (resp. $l :: P$) composed in parallel with N_2 (resp. $l :: Q$), and the states reachable from N_2 (resp. $l :: Q$) composed in parallel with N_1 (resp. $l :: P$). Notice that rules in Table 4 are needed for proving a derivation when one side of the parallel composition is not able to perform a given action. For instance:

$$\frac{l_1 :: \langle 3\rangle_7 \xrightarrow{l_1 \lhd !x@l_2} [\mathbf{0} \mapsto 7] \qquad l_1 :: \langle 4, 5\rangle_7 \xrightarrow{l_1 \lhd !x@l_2} \varnothing}{l_1 :: \langle 3\rangle_7 \parallel l_1 :: \langle 4, 5\rangle_7 \xrightarrow{l_1 \lhd !x@l_2} [l_1 :: \langle 4\rangle_7 \mapsto 7]}$$

Rules (In) and (Read) model synchronisation of parallel components. Within these rules local synchronisations occurring in N_1 (N_2, respectively) are updated in order to take into account the matching tuples available in N_1 (N_2, respectively). This implies that the next state function for such interactions, namely $\mathscr{P}^s \parallel N_2$ ($N_1 \parallel \mathscr{Q}^s$, respectively) must be first "cleaned up" of (total) weight $\oplus(\mathscr{P}^T)$ ($\oplus(\mathscr{Q}^T)$, respectively), relative to N_1 (N_2, respectively) alone. Moreover, the synchronisations between N_1 and N_2 are taken into account. For instance, let $N_1 = l_2 :: \langle 3\rangle_7 \parallel l_1 :: (\mathbf{in}(!x)@l_2, 1).\mathbf{nil}$ and $N_2 = l_2 :: \langle 3\rangle_5$ we have:

$$\frac{\begin{array}{c} N_1 \xrightarrow{l_1:\mathbf{i}(!x:3)@l_2} [\mathbf{0} \mapsto 1] \qquad N_1 \xrightarrow{l_1 \rhd !x 3@l_2} [l_2 :: \langle 3\rangle_7 \mapsto 7] \\ N_1 \xrightarrow{l_1 \lhd !x@l_2} [l_1 :: (\mathbf{in}(!x)@l_2, 1).\mathbf{nil} \mapsto 7] \\ N_1 \xrightarrow{l_1 \lhd 3@l_2} [l_1 :: (\mathbf{in}(!x)@l_2, 1).\mathbf{nil} \mapsto 7] \\ N_2 \xrightarrow{l_1:\mathbf{i}(!x:3)@l_2} \varnothing \qquad N_2 \xrightarrow{l_1 \rhd !x 3@l_2} \varnothing \\ N_2 \xrightarrow{l_1 \lhd !x@l_2} [\mathbf{0} \mapsto 5] \qquad N_2 \xrightarrow{l_1 \lhd 3@l_2} [\mathbf{0} \mapsto 5] \end{array}}{N_1 \parallel N_2 \xrightarrow{l_1:\mathbf{i}(!x:3)@l_2} [l_2 :: \langle 3\rangle_5 \mapsto \frac{7}{12}, l_2 :: \langle 3\rangle_7 \mapsto \frac{5}{12}]}$$

We now present some results that guarantee tractability of the proposed operational semantics. Indeed, we show that each net can only evolve into a finite set of nets. Moreover, we show that if one can derive two different behaviours for N, namely there exists γ such that $N \xrightarrow{\gamma} \mathscr{P}$ and $P \xrightarrow{\gamma} \mathscr{Q}$, then $\mathscr{P} = \mathscr{Q}$.

Theorem 1. *For each N if $N \xrightarrow{\gamma} \mathscr{P}$ then:*

- $\{Q | \mathscr{P}(Q) \neq 0\}$ *is finite;*
- $\oplus \mathscr{P} \leq x$ *for some x.*

Proof □

By induction on the derivation of $N \xrightarrow{\gamma} \mathscr{P}$.

Theorem 2. *For each N and γ if there exist $\mathscr{P}$ and $\mathscr{Q}$ such that $N \xrightarrow{\gamma} \mathscr{P}$ and $N \xrightarrow{\gamma} \mathscr{Q}$ then $\mathscr{P} = \mathscr{Q}$.*

Proof □

By induction on the derivation of $N \xrightarrow{\gamma} \mathscr{P}$.

3.1 Generating Continuous Time Markov Chains

As in previous approaches to enhance calculi with stochastic aspects, see e.g. [20,19,7,8], we use transition relation $\rightarrow$ to associate a CTMC to each StoPKlaim specification. Below we recall the definition of *action-labelled* CTMC (AMC) [15].

Definition 2. *An* action-labelled *CTMC (AMC) $\mathcal{M}$ is a triple $(S, ACT, \mapsto)$ where S is a set of states, ACT is a set of actions, and $\mapsto$ is the transition function, which is a total function from $S \times ACT \times S$ to the set of non-negative real numbers $\mathbb{R}_{\geq 0}$.*

We use the notation $s \overset{\gamma,\lambda}{\longmapsto} s'$ whenever the transition function yields a positive value λ on (s, γ, s'). Transition $s \overset{\gamma,\lambda}{\longmapsto} s'$ intuitively means that the AMC may evolve from state s to s' while performing action γ with an execution time determined by an exponential distribution with rate λ.

The following definition characterises the AMC associated to a StoPKlaim specification.

Definition 3. *Let N be a net, $der(N)$ denotes the smallest set X such that:*

- $N \in X$;
- *if $N_1 \in X$ and $N_1 \xrightarrow{\gamma} \mathscr{P}$ then $\{N' | \mathscr{P}(N') > 0\} \subseteq X$.*

Definition 4. *For* StoPKlaim *specification N such that $der(N)$ is finite, let $\mathsf{AMC}(N) \stackrel{def}{=} (S, ACT, \mapsto)$ with:*

- $S \stackrel{def}{=} der(N)$
- $ACT \stackrel{def}{=} \{\gamma \mid \exists N_1, N_2 \in S, \mathscr{P}.\ N_1 \xrightarrow{\gamma} \mathscr{P} \wedge \mathscr{P}(N_2) > 0\}$
- $N_1 \stackrel{\gamma,\lambda}{\longmapsto} N_2$ *if and only if* $0 < \lambda = \sum_{N_1 \xrightarrow{\gamma} \mathscr{P}} \mathscr{P}(N_2)$.

4 Asynchronous Leader Election Protocol in StoPKlaim

In this Section we use StoPKlaim for specifying a system where n distributed processes have to elect a leader (a uniquely designed process). The problem is resolved by using an adaptation of the *asynchronous leader election protocol* proposed in [17]. In particular, we will show how the proposed calculus can be easily used for analysing and comparing different specifications of a system.

Processes in the system are organized in a ring and can be either *active* or *inactive*. Until a process becomes inactive, it performs the following steps:

1. Chooses 0 or 1 each with a given probability, and sends the choice to the next process.
2. If the process chose 0 and the active process preceding it in the ring chose 1 it becomes inactive and only continues to relay received messages.
3. If it is still active, it sends a counter around the ring to check whether it is the only active process. In that case it becomes the leader, otherwise another round of the algorithm is repeated.

In StoPKlaim the system consists of n nodes each of which hosts the execution of a process. We assume these nodes be $l_0, \ldots, l_{n-1}$. Moreover the process located at l_i precedes the one located at $l_{i+1 \mod n}$ in the ring.

At the beginning, the tuple space located at l_i contains two tuples for modeling process choice ($\langle choice, 0\rangle_{w_1}$ and $\langle choice, 1\rangle_{w_2}$). As the weight of the two tuples coincides ($w_1 = w_2$), we have that each choice has probability 0.5.

The most relevant increment of expressiveness of StoPKlaim with respect to traditional models which are either probabilistic or stochastic is that it allows us to model both *static* and *dynamic* policies for determining the tuple selection probability. In a *static* approach, the probability for a process to select 0 or 1 is fixed and does not change during system evolution. Conversely, in a *dynamic* approach, the probability for a process to select 0 or 1 is not statically defined, but it is modified dynamically depending on run time information such as, e.g., the choices previously done by the process itself or by the previous process in the ring. The dynamic approach is obtained in StoPKlaim simply modifying the weights of the choice tuples $\langle choice, 0\rangle$ and $\langle choice, 1\rangle$ (this can be done removing the two tuples and replacing them with tuples with the same contents but different weights).

We now present a protocol in which a process modifies the probabilities in such a way that at every cycle of the protocol it selects with higher probability a choice different from the choice it previously did.

At the beginning, the system configuration is:

$$\prod_{i=0}^{n-1} \left(l_i :: \langle choice, 0 \rangle_1 \parallel l_i :: \langle choice, 1 \rangle_1 \parallel l_i :: P_i \right)$$

where $\prod_{i=0}^{n-1} N_i$ denotes $N_0 \parallel \cdots \parallel N_{n-1}$ while P_i is defined as follows:

$$\begin{array}{l}
\mathbf{rec}X.\ (\mathbf{in}(choice, !x)@l_i, \lambda_{loc}).(\mathbf{in}(choice, !y)@l_i, \lambda_{loc}). \\
\quad (\mathbf{out}(choice, x:1)@l_i, \lambda_{loc}).(\mathbf{out}(choice, y:3)@l_i, \lambda_{loc}). \\
\quad (\mathbf{out}(prev, x:1)@l_{i+1}, \lambda_{rem}).(\mathbf{in}(prev, !z)@l_i, \lambda_{loc}). \\
\quad \mathbf{if}\ (x = 0\ and\ z = 1)\ \mathbf{then} \\
\qquad \mathbf{rec}Y.(\mathbf{in}(prev, !k)@l_i, \lambda_{loc}).(\mathbf{out}(prev, k:1)@l_{i+1}, \lambda_{rem}).Y \\
\qquad\quad + \\
\qquad\quad (\mathbf{in}(check, !k)@l_i, \lambda_{loc}).(\mathbf{out}(check, k:1)@l_{i+1}, \lambda_{rem}).Y \\
\quad \mathbf{else}\ (\mathbf{out}(check, i:1)@l_{i+1}, \lambda_{rem}).(\mathbf{in}(check, !k)@l_i, \lambda_{loc}). \\
\qquad \mathbf{if}\ (k = i)\ \mathbf{then}\ (\mathbf{out}(leader:1)@l_i, \lambda_{loc}) \\
\qquad \mathbf{else}\ X
\end{array}$$

where, in order to simplify the notation, we use + for the sum modulo n and the *if-then-else* construct even if it is not part of the StoPKlaim syntax (it can be easily encoded using the choice operator). Note that we use two stochastic rates, λ_{loc} for actions on the local tuple space and λ_{rem} for actions on remote tuple spaces.

In order to evaluate the performance of the dynamic protocol with respect to the static one, we have adapted the model checking tool for StoKlaim [8] to deal also with the new language StoPKlaim, and we have used the tool to compute the probability to elect the leader within a predefined amount of time units. In Figure 1 the results of the comparative analysis between the static and the dynamic protocol are depicted (assuming local rate $\lambda_{loc} = 2.5$ and remote rate $\lambda_{rem} = 0.5$). The two protocols have similar performances: only in the interval between 30 and 150 time units the static protocol reports an observable higher

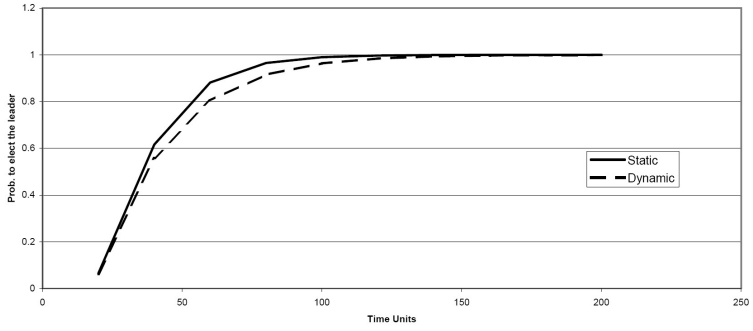

Fig. 1. Comparative analysis of the static and the dynamic protocols

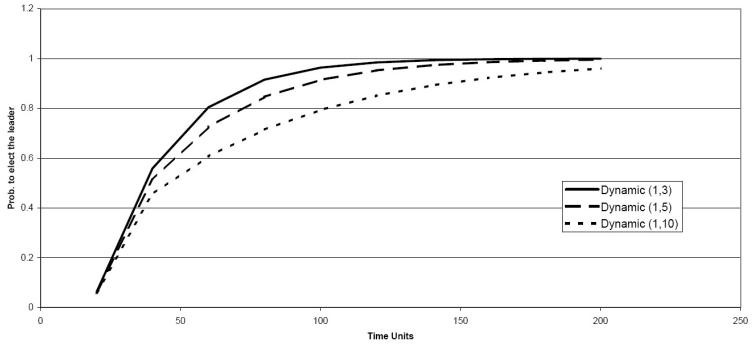

Fig. 2. Dynamic protocol with three different pairs of weights for the choice tuples

probability to complete the protocol. In our analysis we have considered also different dynamic protocols such as a protocol in which a process selects with higher probability a choice different from the one taken by the previous process in the ring at the previous protocol cycle. Rather surprisingly, the performances of the different dynamic protocols essentially coincide. For this reason we report only the analysis of the dynamic protocol formalized above.

As a last analysis, we have considered the dynamic protocol using a different ratio between the two alternative choices. In the description above, indeed, we have used the weights 1 and 3 for the choice tuples. We have repeated the analysis assuming two other pairs of weights: 1 and 5, and 1 and 10. The results of the analysis are depicted in Figure 2. It is immediate to see that an increment of the difference between the weights decreases the performance of the protocol.

5 Conclusions and Related Work

In this paper we have investigated the integration of StoKlaim with PLinda in order to obtain a more expressive modeling language in which the execution time of coordination primitives is modeled by means of exponentially distributed random variables, as in StoKlaim, the choice of the primitive to be executed among conflicting ones is thus resolved by the race condition principle, and the choice of the tuple to be retrieved by a single input/read operation in case of multiple matching tuples is governed by the weight-based probabilistic access policy of PLinda.

We briefly summarise the differences with respect to StoKlaim [11,6] and PLinda [3,4]. The addition of the weight-based probabilistic retrieval mechanism to StoKlaim made it convenient to move from a reduction congruence-based structure of the operational semantics of StoKlaim into a semantics based on transition labels and with no congruence relation.

In particular the adoption of transition systems and definition of the semantics in the style of those in [10] made the usage and involved calculation of rate names that where needed in StoKlaim no longer necessary. Moreover, with respect to

more traditional approaches like that of [16], where multiplicity of identical Markovian transitions is established via implicit counting of the possible ways to infer a Markovian delay, the approach that we use allows for a more explicit definition of the semantics.

With respect to PLinda, which already features a weight-based probabilistic retrieval mechanism similar to the one that we consider here, we mainly added stochastic execution time of coordination operations, hence a race-based choice of them, instead of the non-deterministic choice in PLinda and, as for the case of StoKlaim, we abandoned the reduction congruence-based structure of the operational semantics of PLinda in favor of the semantics style of [10]. The main price to pay for both such extensions is the need of accounting for all possible transitions in the behavior of terms derived by each operational rule (see, e.g., the rule for the parallel operator) in order to correctly account for multiple possible instances of the same transition.

Concerning comparison with other approaches that combine probabilistic mechanisms and Markovain delays we consider the stochastic process algebra EMPA$_{gr}$ [1]. With respect to EMPA$_{gr}$, where the probabilistic choice mechanism can both be performed due to the choice among immediate actions and due to a synchronization between generative Markovian actions and several reactive actions (that are endowed with a weight), the probabilistic choice mechanism of StoPKlaim is only of the latter kind. On the other hand StoPKlaim fully exploits the renormalization capability of weights due to its complex synchronization mechanism based on matching: weights are dynamically re-normalised according to the particular set of matching tuples.

Finally, the approach proposed in this paper could be also applied to other stochastic calculi like, for instance, Stochastic COWS [19], which is a stochastic extension of COWS (Calculus for Orchestration of Web Services) [18]. Currently, the stochastic semantics of COWS, which follows the same approach as [20], is based on proved transition systems.

References

1. Bernardo, M., Bravetti, M.: Performance Measure Sensitive Congruences for Markovian Process Algebras. Theoretical Computer Science 290(1), 117–160 (2003)
2. Bettini, L., Bono, V., De Nicola, R., Ferrari, G., Gorla, D., Loreti, M., Moggi, E., Pugliese, R., Tuosto, E., Venneri, B.: The klaim project: Theory and practice. In: Priami, C. (ed.) GC 2003. LNCS, vol. 2874, pp. 88–150. Springer, Heidelberg (2003)
3. Bravetti, M., Gorrieri, R., Lucchi, R., Zavattaro, G.: Quantitative Information in the Tuple Space Coordination Model. Theoret. Comput. Sci. 346(1), 28–57 (2005)
4. Bravetti, M., Zavattaro, G.: Service Oriented Computing from a Process Algebraic Perspective. Journal of Logic and Algebraic Programming 70(1), 3–14 (2007)
5. De Nicola, R., Ferrari, G., Pugliese, R.: KLAIM: A Kernel Language for Agents Interaction and Mobility. IEEE Transactions on Software Engineering 24(5), 315–329 (1998)

6. De Nicola, R., Katoen, J.-P., Latella, D., Loreti, M., Massink, M.: KLAIM and its stochastic semantics. Technical report, Dipartimento di Sistemi e Informatica, Università di Firenze (2006),
http://rap.dsi.unifi.it/~loreti/papers/TR062006.pdf
7. De Nicola, R., Katoen, J.-P., Latella, D., Loreti, M., Massink, M.: MoSL: A Stochastic Logic for StoKlaim. Tr, ISTI (2006),
http://www1.isti.cnr.it/~Latella/MoSL.pdf
8. De Nicola, R., Katoen, J.-P., Latella, D., Loreti, M., Massink, M.: Model checking mobile stochastic logic. Theoretical Computer Science 382(1), 42–70 (2007)
9. De Nicola, R., Katoen, J.-P., Latella, D., Massink, M.: Towards a logic for performance and mobility. In: Cerone, A., Wiklicky, H. (eds.) Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005). Electronic Notes in Theoretical Computer Science, vol. 153, pp. 161–175. Elsevier, Amsterdam (2006)
10. De Nicola, R., Latella, D., Loreti, M., Massink, M.: MarCaSPiS: a markovian extension of a calculus for services. In: Proc. of SOS 2008 (to appear, 2008)
11. De Nicola, R., Latella, D., Massink, M.: Formal modeling and quantitative analysis of KLAIM-based mobile systems. In: Haddad, H., Liebrock, L., Omicini, A., Wainwright, R., Palakal, M., Wilds, M., Clausen, H. (eds.) APPLIED COMPUTING 2005. Proceedings of the 20th Annual ACM Symposium on Applied Computing, pp. 428–435. Association for Computing Machinery (2005) ISBN 1-58113-964-0
12. Gelernter, D.: Generative Communication in Linda. Communications of the ACM 7(1), 80–112 (1985)
13. Giacalone, A., Jou, C., Smolka, S.: Algebraic reasoning for probabilistic concurrent systems. In: Broy, M., Jones, C. (eds.) Working Conference on Programming Concepts and Methods, IFIP TC 2. North Holland, Amsterdam (1990)
14. Hermanns, H., Herzog, U., Katoen, J.-P.: Process algebra for performance evaluation. Theoret. Comput. Sci. 274(1-2), 43–87 (2002)
15. Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., Siegle, M.: Towards model checking stochastic process algebra. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2000. LNCS, vol. 1945, pp. 420–439. Springer, Heidelberg (2000)
16. Hillston, J.: A compositional approach to performance modelling. Distinguished Dissertation in Computer Science. Cambridge University Press, Cambridge (1996)
17. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Information and Computation 88(1) (1990)
18. Lapadula, A., Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 33–47. Springer, Heidelberg (2007)
19. Prandi, D., Quaglia, P.: Stochastic COWS. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 245–256. Springer, Heidelberg (2007)
20. Priami, C.: Stochastic π-Calculus. The Computer Journal 38(7), 578–589 (1995)
21. Stark, E., Smolka, S.: A complete axiom system for finite-state probabilistic processes. In: Plotkin, G., Stirling, C., Tofte, M. (eds.) Proof, Language and Interaction: Essays in Honour of Robin Milner. MIT Press, Cambridge (2000)