Investigating Fluid-Flow Semantics of Asynchronous Tuple-Based Process Languages for Collective Adaptive Systems

Diego Latella, Michele Loreti, Mieke Massink

Investigating Fluid-flow Semantics of Asynchronous Tuple-based Process Languages for Collective Adaptive Systems [⋆]

Diego Latella[1], Michele Loreti[2], and Mieke Massink[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', CNR, Italy
[2] Università di Firenze & IMT-Lucca, Italy

Abstract. Recently, there has been growing interest in nature-inspired interaction paradigms for Collective Adaptive Systems, for modelling and implementation of adaptive and context-aware coordination, among which the promising pheromone-based interaction paradigm. System modelling in the context of such a paradigm may be facilitated by the use of languages in which adaptive interaction is decoupled in time and space through asynchronous buffered communication, e.g. asynchronous, repository- or tuple-based languages. In this paper we propose a differential semantics for such languages. In particular, we consider an asynchronous, repository based modelling kernel-language which is a restricted version of LINDA, extended with stochastic information about action duration. We provide stochastic formal semantics for both an agent-based view and a population-based view. We then derive an ordinary differential equation semantics from the latter, which provides a fluid-flow deterministic approximation for the mean behaviour of large populations. We show the application of the language and the ODE analysis on a benchmark example of foraging ants.

Keywords: Asynchronous Coordination Languages; Stochastic Process Algebras; Fluid-flow Approximation; Continuous Time Markov Chains.

1 Introduction and Related Work

Collective Adaptive Systems (CAS) are systems typically composed of a large number of heterogeneous agents with decentralised control and varying degrees of complex autonomous behaviour. Agents may be competing for shared resources and, at the same time, collaborate for reaching common goals. The pervasive nature of CAS, together with the importance of the role they play, for instance in the very core of the ICT support for *smart cities*, implies that a serious *a priori* analysis—and, consequently modelling—of the design of any such a system must be performed and that all critical aspects of its behaviour must be carefully investigated before the system is deployed.

[⋆] This research has been partially funded by the EU project QUANTICOL (nr. 600708), and the IT MIUR project CINA.

Recently, there has been growing interest in nature-inspired interaction paradigms for CAS, for enforcing adaptive and context-aware coordination. Among these, those based on the metaphor of pheromones seem promising. System modelling in the context of such a paradigm may be facilitated by the use of languages in which adaptive interaction is decoupled in time and space through asynchronous buffered communication, e.g. tuple-based languages, *a la* LINDA [5]. For systems of limited size, several languages have already been proposed in the literature and have proven useful for modelling—as well as programming—autonomic adaptive coordination. Examples include KLAIM [10], which extends LINDA with, among others, a notion of *space*, the TOTA framework [21], which, additionally, provides for explicit adaptive *tuple propagation* mechanisms and a sort of force field view of tuples, and SCEL [8], where the basic interaction paradigm is enriched with a flexible, *predicate-based* addressing mechanism, with a framework for defining *policies*, and with a notion of tuple-space which is extended to a more general *knowledge*-space. Additionally, quantitative extensions of both KLAIM and SCEL have been developed, namely StoKLAIM [11] and StocS [20], where the quantity of interest is the duration of (the execution of) process actions. Such durations are assumed to be continuous random variables with negative exponential distributions, commonly used in stochastic process algebra [17]. Consequently, each such random variable is fully characterised by its rate, a positive real value that is equal to the inverse of the mean duration of the execution of the action. This choice for action durations gives rise to a Markovian semantics for the languages: the behaviour of each agent of a system is modelled by a continuous time Markov chain (CTMC). The collective behaviour of a system of agents is also modelled by a CTMC, of course obtained as a suitable combination of those of the component agents.

Unfortunately, as soon as the size of the systems under consideration grows, the infamous combinatorial state space explosion problem makes system modelling and analysis essentially unfeasible. On the other hand, one of the key features of CAS is the large size of their component populations. Consequently, *scalability* of modelling—and, most of all, analysis—techniques and tools becomes a *must* in the context of CAS design and development. It is thus essential to develop alternative approaches for modelling systems with *large populations of agents*, possibly based on—and formally linked to—process algebra. In this way, one can try to extend, to such alternative approaches, modelling and analysis techniques which have proven effective for standard stochastic process algebra, such as stochastic model-checking of probabilistic temporal logics. One way to deal with large population systems is the so called *fluid-flow* approach, which consists in computing a deterministic approximation of the mean behaviour of the large population [2]. The first step is to abstract from agent identity and to look only at the *number of agents* in a particular state, for each of the possible states of the agents in the population and at any point in time. Then, a further step is performed by approximating the average values of such numbers by means of a *deterministic, continuous* function of time, which is obtained as the solution of an initial value problem where the set of ordinary differential equations (ODE) is derived from the system model and the initial condition is the initial distribution of the population over the set of local states of the agents. Prominent examples of the fluid-flow approach are the *differential* (i.e. ODE-based) semantics version of the Performance Analysis Process Algebra (PEPA) [23], which we will call ODE-PEPA, Bio-PEPA [7] and, more recently, PALOMA [13]. The advantage of a *fluid-flow* approach is that the transient average behaviour of the system can be analysed orders of magnitude faster than by stochastic simulation, where the mean of a usually large number of simulation traces must be computed. The *fluid-flow* approach is independent of the size of the involved populations, as long as this size is large enough to provide a good deterministic approximation [2].

In this paper we explore the possibility for differential semantics for languages with an asynchronous buffered communication interaction paradigm, e.g. data-repository- /tuple- based ones. We present OdeLinda, a simple experimental language, based on a LINDA-like, asynchronous paradigm, where processes interact only via a data repository by means of **out**, **in** and **read** operations for respectively inserting, withdrawing and reading data values to/from the repository. In particular, we present a *quantitative*, Markovian language; the behaviour of each agent is modelled by a Markov process.

In most stochastic process languages, each action is decorated with its rate, which is typically a constant. In OdeLinda, instead, action rates are allowed to depend on the global state of the complete system; thus they are *functions* from global system states to positive real values, in a similar way as in Bio-PEPA. We provide a formal definition of the Markovian semantics using State-to-Function Labeled Transition Systems (FuTS) [9], an approach that provides for a simple and concise treatment of transition multiplicities—an issue closely related to the CTMC principle of race-condition—and a clean and compact definition of the semantics.

We follow the fluid-flow approach for making the language scalable in order to be able to deal with CAS. We define a population semantics for OdeLinda from which a differential (ODE) semantics is derived, in a similar way as proposed in [13] for PALOMA and in [23] for ODE-PEPA. The interaction paradigm underlying OdeLinda is fundamentally different from those of ODE-PEPA, Bio-PEPA and PALOMA. ODE-PEPA is based on the well-known PEPA process interaction paradigm, with processes synchronising on common, specific activities, Bio-PEPA is based on the chemical-reaction paradigm, whereas PALOMA agents use message multicasting. Additionally, both Bio-PEPA and PALOMA provide some simple means for spatial modelling. Spatial information is currently not incorporated in OdeLinda.

In tuple-based approaches, data repositories are typically *multi-sets* of values and adding/withdrawing a value to/from the repository increases/decreases the multiplicity of that value in the repository. In a “population”-oriented view, this means that the total system population size may change during the computations, i.e. we are dealing with a birth-death type of systems. This is the case for OdeLinda and constitutes another distinguishing feature when compared with e.g. ODE-PEPA. In this respect, our proposal is more similar to sCCP [3], although, from a technical point of view, for the actual definition of the differential semantics we followed the approach used in [23, 13] rather than that presented in [3]. Finally, our work is also related to PALPS [1] and MASSPA [15]. PALPS is a language for ecological models. Only an individual-based semantics is available for PALPS. The language is thus usable only in the specific domain of ecological models and, furthermore, seriously suffers of lack of scalability. MASSPA [15] shares some features with PALOMA, e.g. a multicast-like interaction paradigm; it is lacking a Markovian, individual-based semantics.

It is worth noting that the language we present here is a minimal *kernel* language; we intended to address only the basic issues which arise when defining a differential semantics for tuple-based asynchronous languages. For this reason, operations on data, and in particular *templates* and *pattern-matching* are not considered, so that **in** and **read** operations result into pure synchronisation actions (with or without value consumption, respectively). The unconstrained use of templates and pattern-matching, as well as the use of general operations on data types, could result in an unbounded number of *distinct* values in a model, which, in turn, would require an unbounded number of differential equations in the differential semantics. Consequently, only ground terms are allowed in model specifications. It is worth noting that this does *not* imply that we allow only finite computations or that there are bounds on the multiplicity of each piece of data or on the resulting state spaces. In fact, the number of copies of any given value which can be stored in a repository by means of repeated executions of **out** actions in a computation, by one or more processes, is unbounded (and may be infinite for infinite computations). Anyway, one should also keep in mind that OdeLinda is intended to be a process *modelling*, rather than a programming, language and that differently from most process modelling languages, that, typically, do not provide any feature for dealing with data, offers some means, although primitive, for data storage, withdrawal and retrieval. For the sake of simplicity, we also refrain from considering process spawning, although this would not cause particular problems given that the semantic model we use deals with dynamic population sizes in a natural way. The objective of the present paper is to show that the basic notion of ODE semantics for asynchronous, shared-repository based languages is well founded. Additionally, by revisiting the benchmark example of Foraging Ants, we show that even in the restricted form we present in this paper, OdeLinda can be useful for actual system modelling and analysis.

The present paper is organised as follows: the syntax and Markovian, individual-based semantics of OdeLinda are presented in Section 2; the differential semantics of the language are presented in Section 3. An example of model specification as well as ODE analysis is given in Section 4. Finally, some conclusions and considerations for future work are discussed in Section 5.

2 Syntax and Markovian Semantics of OdeLinda

We recall that the main purpose of this paper is to show the basic principles for the definition of a differential semantics of asynchronous repository-based languages rather than the definition of a complete, high-level, ready-to-use process language. Consequently, the language we present here is a very minimal one, although, as we pointed out in Section 1 it can be used for the effective modelling of typical CAS systems like foraging ants, as we will show in Section 4.

2.1 Syntax

Let $\mathcal{D}$ be a denumerable non-empty set of *data* values, ranged over by $d, d', d_1, \ldots$, $\mathcal{A}$ be set of *actions* with $\mathcal{A} = \mathcal{A}_o \cup \mathcal{A}_i \cup \mathcal{A}_r$, where $\mathcal{A}_o = \{\mathbf{out}(d) \mid d \in \mathcal{D}\}, \mathcal{A}_i = \{\mathbf{in}(d) \mid d \in \mathcal{D}\}, \mathcal{A}_r = \{\mathbf{read}(d) \mid d \in \mathcal{D}\}$, ranged over by $\alpha, \alpha', \ldots$, $\mathcal{P}$ be a denumerable non-empty set of *state constants* (or states), ranged over by $C, C', C_1, \ldots$

A system model is the result of the *parallel composition of agents*, i.e. *processes*, which are finite state machines. Thus the language has the following two level grammar for the sets Agents of agents and Proc of processes:

$$A ::= (R, \mathbf{out}(d)).C \mid (R, \mathbf{in}(d)).C \mid (R, \mathbf{read}(d)).C \mid A + A \qquad P ::= C \mid P \parallel P$$

where for each used constant C there is a definition of the form $C := A$, which, in the sequel, will be written as $C := \sum_{j \in J}(R_j, \alpha_j).C_j$, for some finite index set J, with obvious meaning. In action prefix, $(R, \alpha).\_$, R is the name of a *rate function* under the scope of a suitable definition $R := E$; E is a numerical expression where the special operator $\#C$ can be used which, for state name C, yields the number of agents which are in state C in the current global system state. We will refrain from giving further details on the syntax of expressions E.

A *process definition* is the collection of definitions for the states of the process. A *system state* is a pair (P, D) where the set Reps of data repositories D is defined according to the following grammar:

$$D ::= \quad \langle\rangle \quad \mid \quad \langle d \rangle \quad \mid \quad D|D$$

The language of expressions E for rate function definitions is extended with $\#d$, for values $d \in \mathcal{D}$, with the obvious meaning. A *system* (model) *specification* is composed of the set of definitions for its processes, the set of definitions for the rate functions used therein, and an initial global state (P_0, D_0). It is required that for each state in the system specification there is exactly one definition. For the sake of simplicity, in the present paper we require that for all $i \in I$ and $x \in \{o, i, r\}$, if $\alpha_{ij} \in \mathcal{A}_x$ for some $j \in J_i$, then $\alpha_{ih} \in \mathcal{A}_x$ for all $h \in J_i$ (no mixed choice) and that for $C \neq C'$, if $C := \sum_{j \in J}(R_j, \alpha_j).C_j$ and $C' := \sum_{j \in J'}(R'_j, \alpha'_j).C'_j$ are both state definitions appearing in the system definition then $\{\alpha_j\}_{j \in J} \cap \{\alpha'_j\}_{j \in J'} \neq \emptyset$ implies $\{\alpha_j\}_{j \in J} = \{\alpha'_j\}_{j \in J'}$ (in order not to incur in the possibility of circular definitions in the ODE). In the sequel we let $\mathcal{S}$ denote the set of global system states.

Agents:

Reader	= (RA, **in**(a)).Comp + (RB, **in**(b)).Comp
Comp	= (RR, **read**(r)).Reader
AWriter	= (WA, **out**(a)).AWriter
BWriter	= (WB, **out**(b)).AWriter

Rate Functions:

RA	$= 10 \cdot \#\text{Reader} \cdot \#\text{a}$
RB	$= 5 \cdot \#\text{Reader} \cdot \#\text{b}$
RR	$= 10 \cdot \#\text{Comp} \cdot \#\text{r}$
WA	$= 9 \cdot \#\text{AWriter}$
WB	$= 4 \cdot \#\text{BWriter}$

Fig. 1. A simple model of Readers and Writers

As a simple running example we consider the specification of a readers/writers model given in Figure 1, where two kinds of writers are considered—those writing messages of type a and those writing messages of type b—and readers perform some computation using some resources r before reading the next item, modelled by synchronisation on r—with the following initial state[3]

$$(\mathit{Reader}[10000] \parallel \mathit{AWriter}[5000] \parallel \mathit{BWriter}[5000], \langle a\rangle[5000]|\langle b\rangle[5000])|\langle r\rangle[1000]).$$

2.2 Stochastic semantics

The stochastic semantics are given in Figure 2 using the FuTS framework [9], that is an alternative to the classical approach, based on Labelled Transition Systems (LTS). In LTS, a transition is a triple (s, α, s') where s and α are the source state and the label of the transition, respectively, while s' is the target state reached from s via a transition labeled with α. In FuTS, a transition is a triple of the form $(s, \alpha, \mathscr{F})$. The first and second component are the source state and the label of the transition, as in LTS, while the third component $\mathscr{F}$ is a *continuation function* (or simply a *continuation* in the sequel), which associates a value from an appropriate semiring with each state s'. In the case of Markovian process algebra, the relevant semiring is that of non-negative real numbers. If $\mathscr{F}$ maps s' to 0, then state s' cannot be reached from s via this transition. A positive value for state s' represents the rate for the jump of the system from s to s'. Any FuTS over $\mathbb{R}_{\geq 0}$ uniquely defines a CTMC, which can obviously be built by successive application of the continuations to the set of states. Below we recall the main notions on FuTS. The reader interested in further details is referred to [9].

[3] We use the standard notational convention that $P[n]$ means n instances of P in parallel: $P \parallel P \parallel \ldots \parallel P$. We extend it to tuples in the obvious way.

Given a denumerable non-empty set V, we let $\mathbf{FS}(V, \mathbb{R}_{\geq 0})$ denote the class of finitely supported[4] functions from V to $\mathbb{R}_{\geq 0}$. For $v_1, \ldots, v_n$ in set V and $r_1, \ldots, r_n \in \mathbb{R}_{\geq 0}$, we let $[v_1 \mapsto r_1, \ldots, v_n \mapsto r_n]$ in $\mathbf{FS}(V, \mathbb{R}_{\geq 0})$ denote the function mapping v_i to r_i, for $i = 1, \ldots n$, and any other $v \in V \setminus \{v_1, \ldots, v_n\}$ to 0; the degenerate case $[]$ denotes the constant function yielding 0 everywhere. For $v \in V$, $\mathcal{X}\, v$ denotes the function $[v \mapsto 1]$. For functions $\mathscr{F}_1, \mathscr{F}_2 \in \mathbf{FS}(V, \mathbb{R}_{\geq 0})$ we let $(\mathscr{F}_1 + \mathscr{F}_2) \in \mathbf{FS}(V, \mathbb{R}_{\geq 0})$ be defined as $(\mathscr{F}_1 + \mathscr{F}_2)\, v = (\mathscr{F}_1\, v) + (\mathscr{F}_2\, v)$ and we extend $(\mathscr{F}_1 + \mathscr{F}_2)$ to the n-ary version $\sum_{j \in J} \mathscr{F}_j$, in the obvious way, for finite index set J. For $r \in \mathbb{R}$ we let $\mathscr{F}/r \in \mathbf{FS}(V, \mathbb{R}_{\geq 0})$ be the defined as $(\mathscr{F}/r)\, v = (\mathscr{F}\, v)/r$ if $r \neq 0$ and $(\mathscr{F}/r)\, v = 0$ otherwise. We let $\oplus\mathscr{F}$ be defined as $\oplus\mathscr{F} = \sum_{v \in V} (\mathscr{F}\, v)$; note that $\oplus\mathscr{F}$ is finite, and thus well-defined, for $\mathscr{F} \in \mathbf{FS}(V, \mathbb{R}_{\geq 0})$. We recall standard structural congruence $\equiv$ on Reps, with $D|\langle\rangle \equiv D, D_1|D_2 \equiv D_2|D_1, (D_1|D_2)|D_3 \equiv D_1|(D_2|D_3)$. In the sequel, when dealing with data repositories, we will implicitly assume them modulo $\equiv$. For the sake of notational simplicity we will keep $D, D' \ldots$ in the notation (but actually the representatives of their equivalence classes are intended). A similar structural congruence $\equiv$ is assumed for processes, with $P_1 \parallel P_2 \equiv P_2 \parallel P_1, (P_1 \parallel P_2) \parallel P_3 \equiv P_1 \parallel (P_2 \parallel P_3)$, as well as similar conventions concerning notation.

For function $\mathscr{P}$ in $\mathbf{FS}(\textsc{Proc}_{/\equiv}, \mathbb{R}_{\geq 0})$ and $\mathscr{D}$ in $\mathbf{FS}(\textsc{Reps}_{/\equiv}, \mathbb{R}_{\geq 0})$ the notation $(\mathscr{P}, \mathscr{D})$ defines a function in $\mathbf{FS}(\mathcal{S}_{/\equiv}, \mathbb{R}_{\geq 0})$ as follows: for system state $(P, D) \in \mathcal{S}_{/\equiv}$, we have $(\mathscr{P}, \mathscr{D})(P, D) = (\mathscr{P}\, P) \cdot (\mathscr{D}\, D)$, where $\cdot$ denotes product in $\mathbb{R}_{\geq 0}$. For each rate function definition $R = E$, we consider the function $R : \mathcal{S}_{/\equiv} \to \mathbb{R}_{\geq 0}$ defined in the following. For all $(P, D) \in \mathcal{S}_{/\equiv}$ $R(P, D) = [\![E]\!]_{(P,D)}$, where $[\![E]\!]_{(P,D)}$ denotes the value of expression E in the current global state (P, D). Obviously $[\![\#C]\!]_{(C,D)} = 1$ and $[\![\#C]\!]_{(P_1 \parallel P_2, D)} = [\![\#C]\!]_{(P_1,D)} + [\![\#C]\!]_{(P_2,D)}$. The definition for $[\![\#d]\!]_{(P,D)}$ is similar.

The continuation summation in Rule (PA) takes care of multiple alternatives with the same action and rate function. Different choices for functions $\mathcal{R}_o$, $\mathcal{R}_i$, $\mathcal{R}_r$ in rules (IN), (OUT) and (READ) give rise to different interaction policies. For instance, for a TIPP-like synchronisation policy, assuming each rate function definition be of the form $R := k_R$, for $k_R \in \mathbb{R}_{>0}$, one can let $\mathcal{R}_o(P, D, \mathscr{P}, \mathscr{D}, R) = \mathcal{R}_i(P, D, \mathscr{P}, \mathscr{D}, R) = \mathcal{R}_r(P, D, \mathscr{P}, \mathscr{D}, R) = R(P, D) \cdot (\mathscr{P}, \mathscr{D})$. Similarly, for a PEPA-like interaction paradigm, assuming again each rate function be a constant, we get $\mathcal{R}_o(P, D, \mathscr{P}, \mathscr{D}, R) = R(P, D) \cdot (\mathscr{P}, \mathscr{D})$, and $\mathcal{R}_i(P, D, \mathscr{P}, \mathscr{D}, R) = \mathcal{R}_r(P, D, \mathscr{P}, \mathscr{D}, R) = R(P, D) \cdot \frac{\min\{\oplus\mathscr{P}, \oplus\mathscr{D}\}}{\oplus\mathscr{P} \cdot \oplus\mathscr{D}} (\mathscr{P}, \mathscr{D})$. In this paper we choose $\mathcal{R}_o = \mathcal{R}_i = \mathcal{R}_r = \mathcal{R}$ where

$$\mathcal{R}(P, D, \mathscr{P}, \mathscr{D}, R) = R(P, D) \cdot (\frac{\mathscr{P}}{\oplus\mathscr{P}}, \frac{\mathscr{D}}{\oplus\mathscr{D}}).$$

The idea is that the rate of the action is the full responsibility of the modeller, being equal to $R(P, D)$; in fact $(\frac{\mathscr{P}}{\oplus\mathscr{P}}, \frac{\mathscr{D}}{\oplus\mathscr{D}})(P', D')$ is equal to 1 if (P', D') is reachable in one transition from (P, D) and 0, if it is not.

[4] A function $f : V \to \mathbb{R}_{\geq 0}$ has finite support if and only if the set $\{v \in V \mid f\, v \neq 0\}$ is finite. In this paper we often use Currying in function application.

$$\text{PA:}\ \frac{C{:=}\sum_{j\in J}(R_j,\alpha_j).C_j}{C \overset{R,\alpha}{\rightarrowtail} \sum_{\{h\in J,\alpha_h=\alpha\wedge R_h=R\}}[C_h\mapsto 1]} \qquad \text{DI:}\ \frac{}{\langle d\rangle \overset{\mathbf{in}(d)}{\rightarrowtail} [\langle\rangle\mapsto 1]}$$

$$\text{DR:}\ \frac{}{\langle d\rangle \overset{\mathbf{read}(d)}{\rightarrowtail} [\langle d\rangle\mapsto 1]} \qquad \text{DN}_1\text{:}\ \frac{}{\langle\rangle \overset{\alpha}{\rightarrowtail} []} \qquad \text{DN}_2\text{:}\ \frac{\alpha\notin\{\mathbf{in}(d),\mathbf{read}(d)\}}{\langle d\rangle \overset{\alpha}{\rightarrowtail} []}$$

$$\text{PP:}\ \frac{P_1 \overset{R,\alpha}{\rightarrowtail} \mathscr{P}_1 \qquad P_2 \overset{R,\alpha}{\rightarrowtail} \mathscr{P}_2}{P_1|P_2 \overset{R,\alpha}{\rightarrowtail} \mathscr{P}_1|(\mathcal{X}\,P_2)+(\mathcal{X}\,P_1)|\mathscr{P}_2} \qquad \text{DP:}\ \frac{D_1 \overset{\alpha}{\rightarrowtail} \mathscr{D}_1 \qquad D_2 \overset{\alpha}{\rightarrowtail} \mathscr{D}_2}{D_1|D_2 \overset{\alpha}{\rightarrowtail} \mathscr{D}_1|(\mathcal{X}\,D_2)+(\mathcal{X}\,D_1)|\mathscr{D}_2}$$

$$\text{OUT:}\ \frac{P \overset{R,\mathbf{out}(d)}{\rightarrowtail} \mathscr{P}}{(P,D) \overset{\mathbf{out}(d)}{\rightarrowtail} \mathcal{R}_o(P,D,\mathscr{P},\mathcal{X}(D|\langle d\rangle),R)}$$

$$\text{IN:}\ \frac{D \overset{\mathbf{in}(d)}{\rightarrowtail} \mathscr{D}\ P \overset{R,\mathbf{in}(d)}{\rightarrowtail} \mathscr{P}}{(P,D) \overset{\mathbf{in}(d)}{\rightarrowtail} \mathcal{R}_i(P,D,\mathscr{P},\mathscr{D},R)} \qquad \text{READ:}\ \frac{D \overset{\mathbf{read}(d)}{\rightarrowtail} \mathscr{D} \quad P \overset{R,\mathbf{read}(d)}{\rightarrowtail} \mathscr{P}}{(P,D) \overset{\mathbf{read}(d)}{\rightarrowtail} \mathcal{R}_r(P,D,\mathscr{P},\mathscr{D},R)}$$

Fig. 2. FuTS semantics of the process language with tuple creation

3 Differential Semantics of OdeLinda

In this section we define the differential semantics for the language introduced in Sect. 2. We follow a similar approach as in [13, 23]: we first define a population semantics for the language and then we define the differential semantics by means of deriving, from the population semantics, suitable ODEs for the mean-field model.

3.1 Population semantics

Assume we are given a system specification where $\{C_1, \dots C_s\}$ is the set of all states of all processes and $\{d_1, \dots d_t\}$ is the set of all data values textually occurring in the specification. Given a global system state (P, D) we consider the corresponding vector $\mathbf{X} = (x_1, \dots, x_m)$ of counting variables such that, for $i = 1, \dots, s$, x_i records the number of agents in P which are in (local) state C_i, and for $i = s+1, \dots, m = s+t$, x_i records the number of instances of d_{i-s} in D. Clearly, every transition at the single agent level corresponds to a change in the value of $\mathbf{X}$, i.e. a *population-based* transition. In order to formalise how single agent transitions induce population-based transitions, let (P', D') and (P'', D'') be two global system states with $P' = C'_1 \parallel \dots \parallel C'_{s'}$, $P'' = C''_1 \parallel \dots \parallel C''_{s''}$, $D' = d'_1|\dots|d'_{t'}$, and $D'' = d''_1|\dots|d''_{t''}$ and, with reference to the given system specification, define the *update vector* $\boldsymbol{\delta}$ in the usual way[5]:

[5] $\mathbf{1}\{C = C'\}$ is equal to 1 if $C = C'$ and to 0 otherwise.

$\boldsymbol{\delta}((P'', D''), (P', D')) = (\delta_1, \ldots, \delta_m)$ with

$$\delta_i = \begin{cases} \sum_{j=1}^{s''} \mathbf{1}\{C''_j = C_i\} - \sum_{j=1}^{s'} \mathbf{1}\{C'_j = C_i\}, \text{ for } i = 1, \ldots, s \\ \\ \sum_{j=1}^{t''} \mathbf{1}\{d''_j = d_i\} - \sum_{j=1}^{t'} \mathbf{1}\{d'_j = d_i\}, \text{ for } i = s+1, \ldots, m \end{cases}$$

With the definition of the update vector in place we can easily define the population-based transitions using the following rule:

$$\frac{(P, D) \overset{\alpha}{\rightarrowtail} (\mathscr{P}, \mathscr{D}) \quad r(P, D) = (\mathscr{P}, \mathscr{D})(P', D') > 0}{\mathbf{X} \overset{\alpha, r(P,D)}{\rightarrow} \mathbf{X} + \boldsymbol{\delta}((P', D'), (P, D))}$$

Using the above procedure, for any system specification we can derive a population-based CTMC (PCTMC) [2]. Such a PCTMC is defined as the tuple $(\mathbf{X}, \mathbb{Z}^m, \mathcal{T}, \mathbf{x}_0)$, where, :

- $\mathbf{X} = (x_1, \ldots, x_m)$ is the state vector, where, for $i = 1, \ldots, s$ element x_i is the count of agents in state C_i and, for $i = s+1, \ldots, m = s+t$ it counts the number of instances of d_{i-s};
- $\mathcal{T}(\mathbf{X}) = \{\tau_1, \ldots, \tau_h\}$ is the set of population-based transitions enabled in state $\mathbf{X}$. Each transition τ is associated with a update vector $\boldsymbol{\delta}_\tau$ and a rate $r_\tau(\mathbf{X}) = \sum\{r \mid \mathbf{X} \overset{\alpha, r}{\rightarrow} \mathbf{X} + \boldsymbol{\delta}_\tau \text{ for some } \alpha\}$;
- $\mathbf{x}_0 \in \mathbb{Z}^m$ is the initial state of the PCTMC.

3.2 Mean-field model

The dynamics of the above PCTMC is as follows: if the PCTMC is currently in state $\mathbf{X}$, then, every $1/r_\tau(\mathbf{X})$ time units, on average, a change in the population level of some agents and data items $\boldsymbol{\delta}_\tau$ occurs. We can approximate such a discrete change in a continuous way so that for small finite time interval Δt the change in the population level is

$$\mathbf{X}(t + \Delta t) = \mathbf{X}(t) + r_\tau(\mathbf{X}(t)) \cdot \Delta t \cdot \boldsymbol{\delta}_\tau$$

from which, for $\Delta t \to 0$, we get the ODE $\frac{\mathrm{d}\mathbf{X}(t)}{\mathrm{d}t} = r_\tau(\mathbf{X}(t)) \cdot \boldsymbol{\delta}_\tau$. Taking all enabled transitions into account the ODE describing the approximated transient evolution of the complete population-level system dynamics is given by the initial value problem:

$$\frac{\mathrm{d}\mathbf{X}(t)}{\mathrm{d}t} = \sum_{k=1}^{h} r_{\tau_k}(\mathbf{X}(t)) \cdot \boldsymbol{\delta}_{\tau_k} \qquad \text{with } \mathbf{X}(0) = \mathbf{x}_0$$

for large populations and under suitable scalability assumptions (on the rate functions); the interested reader is referred to [2] for the technical details.

With reference to our running example of Figure 1 we get the equations of Figure 3. Note that there is no dynamics for the writer processes in this example, since each of these agents has just a single state (and a self-loop). Similarly for the resource r.

$$
\begin{aligned}
\frac{d\,\mathrm{Reader}(t)}{dt} &= 10 \cdot \mathrm{r}(t) \cdot \mathrm{Comp}(t) - (10 \cdot \mathrm{a}(t) + 5 \cdot \mathrm{b}(t)) \cdot \mathrm{Reader}(t) \\
\frac{d\,\mathrm{Comp}(t)}{dt} &= (10 \cdot \mathrm{a}(t) + 5 \cdot \mathrm{b}(t)) \cdot \mathrm{Reader}(t) - 10 \cdot \mathrm{r}(t) \cdot \mathrm{Comp}(t) \\
\frac{d\,\mathrm{a}(t)}{dt} &= 9 \cdot \mathrm{AWriter(t)} - 10 \cdot \mathrm{a}(t) \cdot \mathrm{Reader}(t) \\
\frac{d\,\mathrm{b}(t)}{dt} &= 4 \cdot \mathrm{BWriter(t)} - 5 \cdot \mathrm{a}(t) \cdot \mathrm{Reader}(t)
\end{aligned}
$$

Fig. 3. ODE for the simple model of Readers and Writers of Figure 1

4 Example - Foraging Ants

As an example, we revisit a somewhat simplified model of a colony of foraging ants inspired by earlier work in the literature [14, 12, 22]. The ants initially reside at a *Nest* and will move between the *Nest* and a *Food* site. There are two, bidirectional, paths connecting the *Nest* to the *Food* site (and vice-versa), the *Fast* path and the *Slow* path. Each path is composed by a finite sequence of (path) stages: the number ℓ_F of stages of the *Fast* path is smaller than the number ℓ_S of stages of the *Slow* path. The average time it takes an ant to traverse a stage is the same for each stage, regardless of whether it is situated on the *Slow* or the *Fast* path; such traversal times are modelled by exponentially distributed random variables. The situation is depicted in Fig. 4 where FP_j stands for the j-th of the ℓ_F stages of the *Fast* path and SP_j stands for the j-th of the ℓ_S stages of the *Slow* path, for $\ell_F < \ell_S$. A model for foraging ants is specified in

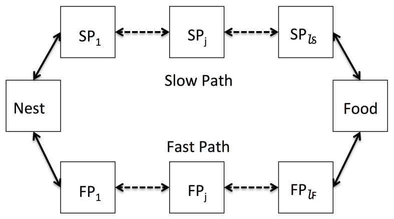

Fig. 4. Schematic of a *Fast* Path and a *Slow* Path for ants; FP_j represents stage j of the Fast Path and SP_j represents stage j of the Slow Path; $\ell_F < \ell_S$ is assumed.

Fig. 5. The set of data values occurring in the model specification is the finite set $(\bigcup_{j=1}^{\ell_F}\{\mathrm{Phe@FP}_j\}) \cup (\bigcup_{j=1}^{\ell_S}\{\mathrm{Phe@SP}_j\})$ where tuple $\langle\mathrm{Phe@FP}_j\rangle$ ($\langle\mathrm{Phe@SP}_j\rangle$, respectively) represents a unit of pheromone in stage j of the *Fast* path (*Slow* path, respectively). There are two process types, one modelling an ant and one modelling the expiration, i.e. decay, of pheromones; the (finite) set of states is as follows:

$$
\begin{aligned}
&\{\mathrm{Ant@Nest}, \mathrm{Ant@Food}\} \cup \\
&(\textstyle\bigcup_{j=1}^{\ell_F}\{\mathrm{AntToFood@FP}_j\}) \cup (\bigcup_{j=1}^{\ell_S}\{\mathrm{AntToFood@SP}_j\}) \cup \\
&(\textstyle\bigcup_{j=1}^{\ell_F}\{\mathrm{AntToNest@FP}_j\}) \cup (\bigcup_{j=1}^{\ell_S}\{\mathrm{AntToNest@SP}_j\}) \cup
\end{aligned}
$$

$(\bigcup_{j=1}^{\ell_F}\{\mathrm{ExpPhe@FP}_j\}) \cup (\bigcup_{j=1}^{\ell_S}\{\mathrm{ExpPhe@SP}_j\})$.

State Ant@Nest (Ant@Food, respectively) represents an ant at the *Nest* (at the *Food* site, respectively). State AntToFood@FP$_j$ (AntToFood@SP$_j$, respectively) represents an ant in stage j of the *Fast* (*Slow*, respectively) path, when travelling from the *Nest* to the *Food*. State AntToNest@FP$_j$ (AntToNest@SP$_j$, respectively) represents an ant in stage j of the *Fast* (*Slow*, respectively) path, when travelling from the *Food* to the *Nest*. For the sake of simplicity, in this model, once an ant leaves the *Nest*, it can only proceed to the *Food* and then come back to the *Nest* (i.e. an ant cannot change its mind half-way a path or get stuck there). This is common in foraging ants models (see [22] and references therein). Finally, processes ExpPhe@FP$_j$ and ExpPhe@SP$_j$ are used for modelling pheromone decay. The definitions of rate functions NFF, NFS, NFF$_j$, NFS$_j$, PHF, and PHS are given below, where parameters k, m and p will be discussed later on in this section:

$$\begin{aligned}
\mathrm{NFF} &= \tfrac{(k+\#\mathrm{Phe@FP}_1)^2}{(k+\#\mathrm{Phe@FP}_1)^2+(k+\#\mathrm{Phe@SP}_1)^2} \cdot \#\mathrm{Ant@Nest}\\
\mathrm{NFS} &= \tfrac{(k+\#\mathrm{Phe@SP}_1)^2}{(k+\#\mathrm{Phe@FP}_1)^2+(k+\#\mathrm{Phe@SP}_1)^2} \cdot \#\mathrm{Ant@Nest}\\
\mathrm{NFF}_j &= m \cdot \#\mathrm{AntToFood@FP}_j\\
\mathrm{NFS}_j &= m \cdot \#\mathrm{AntToFood@SP}_j\\
\mathrm{PHF} &= p \cdot \#\mathrm{ExpPhe@FP}_j\\
\mathrm{PHS} &= p \cdot \#\mathrm{ExpPhe@SP}_j
\end{aligned}$$

The expressions for the definitions of NFF and NFS are written in accordance with results from experimental studies on colonies of Argentine ants, as discussed in [14, 12, 22]. The definition of functions FNF, FNS, FNF$_j$, and FNS$_j$ are similar to those of NFF, NFS, NFF$_j$ and NFS$_j$ due to the symmetry of the model.

In the following we present some analysis results for two specific instantiations of the model and its parameters. We consider a model where the *Fast* path is composed of two stages while the *Slow* path is seven stages long, i.e. $\ell_F = 2$ and $\ell_S = 7$.

We first assume that pheromones do not decay: they accumulate in the path stages so that their total amount grows larger and larger. This is achieved by setting the decay rate p to zero. The value chosen for k is 10, while the ants rate of movement m from one path stage to the next one is set to 0.1. Fig. 6 shows the solution of the equations for the first 500 time units for an initial number of 1000 ants in the *Nest*, while no ants are assumed present in any other path stage, neither at the *Food* site, initially. One unit of pheromone is assumed to be present at time 0 at the stages of the *Slow* and the *Fast* paths closest to the *Nest*. Fig. 6 (left) shows that there is a quick drop in the number of ants at the *Nest* and that for a brief time frame of about 50 time units the cumulative number of ants on the *Slow* path is actually higher than that on the *Fast* path. This situation changes rapidly when ants start to return from the *Food* to the *Nest* providing implicitly feedback to the system by reinforcing the pheromone

$$\begin{array}{lll}
\mathrm{Ant@Nest} & = (\mathrm{NFF},\ \mathbf{read}(\mathrm{Phe@FP}_1)).\mathrm{AntToFood@FP}_1\ + & \\
 & \ (\mathrm{NFS},\ \mathbf{read}(\mathrm{Phe@SP}_1)).\mathrm{AntToFood@SP}_1 & \\
\mathrm{AntToFood@FP}_j & = (\mathrm{NFF}_j,\ \mathbf{out}(\mathrm{Phe@FP}_j)).\mathrm{AntToFood@FP}_{j+1} & j = 1 \ldots \ell_F - 1 \\
\vdots & & \\
\mathrm{AntToFood@FP}_{\ell_F} & = (\mathrm{NFF}_{\ell_F},\ \mathbf{out}(\mathrm{Phe@FP}_{\ell_F})).\mathrm{Ant@Food} & \\
\mathrm{AntToFood@SP}_j & = (\mathrm{NFS}_j,\ \mathbf{out}(\mathrm{Phe@SP}_j)).\mathrm{AntToFood@SP}_{j+1} & j = 1 \ldots \ell_S - 1 \\
\vdots & & \\
\mathrm{AntToFood@SP}_{\ell_S} & = (\mathrm{NFS}_{\ell_F},\ \mathbf{out}(\mathrm{Phe@SP}_{\ell_F})).\mathrm{Ant@Food} & \\
\mathrm{Ant@Food} & = (\mathrm{FNF}_{\ell_F},\mathbf{read}(\mathrm{Phe@FP}_{\ell_F})).\mathrm{AntToNest@FP}_{\ell_F}\ + & \\
 & \ (\mathrm{FNS}_{\ell_S},\mathbf{read}(\mathrm{Phe@SP}_{\ell_S})).\mathrm{AntToNest@SP}_{\ell_S} & \\
\mathrm{AntToNest@FP}_{\ell_F - j} & = (\mathrm{FNF}_{\ell_F - j},\mathbf{out}(\mathrm{Phe@FP}_{\ell_F - j})).\mathrm{AntToNest@FP}_{\ell_F-(j+1)} & j = 1 \ldots \ell_F - 1 \\
\vdots & & \\
\mathrm{AntToNest@FP}_1 & = (\mathrm{FNF}_1,\mathbf{out}(\mathrm{Phe@FP}_1)).\mathrm{Ant@Nest} & \\
\mathrm{AntToNest@SP}_{\ell_S - j} & = (\mathrm{FNS}_{\ell_S - j},\ \mathbf{out}(\mathrm{Phe@SP}_{\ell_F - j})).\mathrm{AntToNest@SP}_{\ell_F-(j+1)} & \\
 & & j = 1 \ldots \ell_S - 1 \\
\vdots & & \\
\mathrm{AntToNest@SP}_1 & = (\mathrm{FNS}_1,\ \mathbf{out}(\mathrm{Phe@SP}_1)).\mathrm{Ant@Nest} & \\
 & & \\
\mathrm{ExpPhe@FP}_j & = (\mathrm{PHF},\ \mathbf{in}(\mathrm{Phe@FP}_j)).\mathrm{ExpPhe@FP}_j & j = 1 \ldots \ell_F \\
\mathrm{ExpPhe@SP}_j & = (\mathrm{PHS},\ \mathbf{in}(\mathrm{Phe@SP}_j)).\mathrm{ExpPhe@SP}_j & j = 1 \ldots \ell_S
\end{array}$$

Fig. 5. OdeLinda model for foraging ants

trace on the *Fast* path. This leads to a rather quick convergence of ants on the *Fast* path. The cumulative amount of pheromones on the *Fast* and the *Slow* path is shown in Fig. 6 (right).

Fig. 7 shows the results for a variant of the model where pheromone decays with constant rate $p = 0.03$. The evolution of both the cumulative number of ants in various locations and the amount of pheromone on the paths is shown over a time interval of 500 time units. Also in this case the ants converge on the *Fast* path and they are doing so in shorter time than in the case without decay of pheromones. Fig. 7 (left) has been obtained using Octave[6] for solving the ODE for the model specification. Fig. 7 (right) shows the results obtained via stochastic simulation for the same model with 1000 ants taking the average over 100 runs[7].

We close this section showing the application of the mean field model-checker FlyFast [19] on the foraging ants example. Fluid model-checking techniques have recently been proposed as scalable techniques for the verification of properties of one (or a few) agents in the context of large populations [4]. These techniques are based on differential semantics, or on difference equations, when considering their discrete time counterparts, as is the case for FlyFast. The input language of the model-checker does not support the specification of models with dynamic population size, but if we can assume sufficiently large upper bounds on the

[6] See for information on Octave http://www.octave.org. Version 3.4.0 was used.

[7] Experiments were conducted with a 1.8 GHz Core i7 Intel processor and 4 GB RAM running Mac OS X 10.7.5.

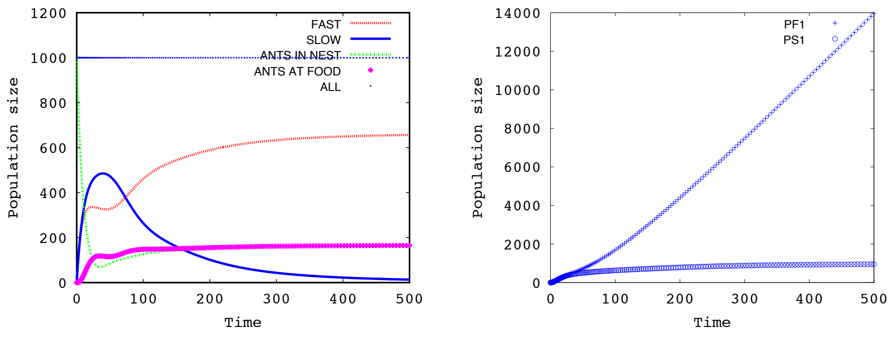

Fig. 6. Evolution over the first 500 time units of the number of ants in the *Nest*, at the *Food* and on the *Fast* and the *Slow* path (left) and the amounts of pheromones on the start of the short (PF1) and long (PS1) path (right).

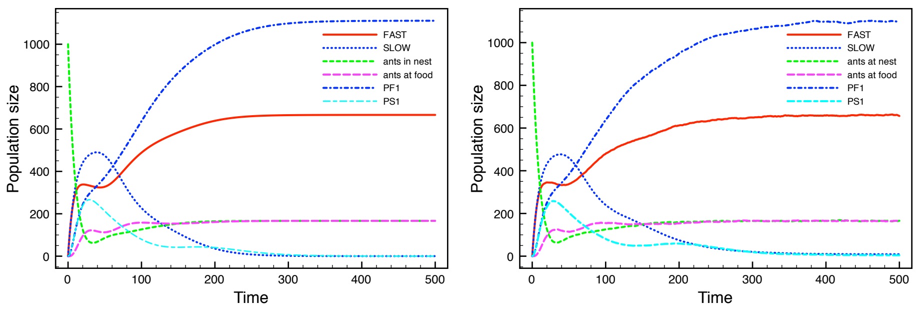

Fig. 7. Evolution over the first 500 time units of the system allowing decay of pheromones with rate 0.03. Solution of the differential equations (left) and stochastic simulation average over 100 runs for a model with 1000 ants (right).

sizes of the data sub-populations for the time horizon of interest[8], it is rather straightforward to translate the model of foraging ants shown in Fig. 5 into such a language, modelling data by two-state (i.e. "present" and "absent") processes and using an appropriate scaling of rates to turn the stochastic model into an equivalent probabilistic one [18]. We briefly illustrate the results for two properties for the ants model in Fig. 8. Property A shows how the probability of an ant in the nest to move to the short path within 30 time units changes over time due to the pheromones left behind by other ants. Property B shows the probability to reach a system state within t time units, where t ranges from 0 to 500, in which an ant in the nest moves to the short path within 30 time units with a probability of more than 0.95. Both properties can be expressed using

[8] This can be checked using the result of an ODE analysis or simulation.

the standard PCTL logic, a probabilistic extension of CTL [16]. Model-checking times for property A is 10 ms, whereas that of property B is 41,047 ms.

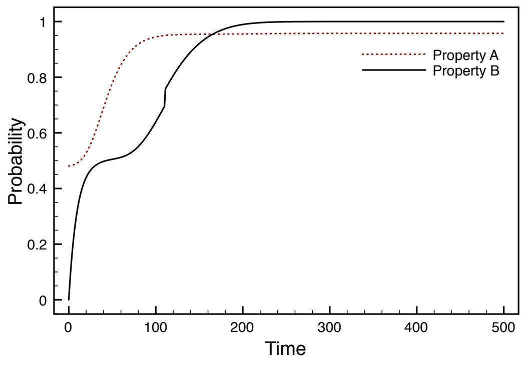

Fig. 8. Mean field model-checking results for properties A and B.

The purpose of the foraging ants example is to illustrate the OdeLinda language and mean field analysis approach for asynchronous, tuple based languages. Results better matching those of the original experiments in [14, 12] can be obtained by a somewhat more complicated model in which ants leave the nest at a constant rate and in which the length of the paths and the constant traversal times are more accurately modelled by adding further path stages on each path implicitly using an Erlang distribution with more stages to approximate the constant traversal times. We omitted this here for the sake of simplicity.

5 Conclusions and Future Work

In this paper we have provided a differential semantics for languages with an asynchronous buffered communication interaction paradigm, e.g. data-repository-/tuple-based ones. In particular, we have defined an individual-based Markovian as well as population based differential semantics for OdeLinda, a simple data-repository-based language. As example of use of the language we have shown a benchmark model of Foraging Ants and some results of its ODE-based analysis. There are several lines of research we plan to follow for moving from a simple experimental kernel language like OdeLinda to a complete, full fledged population modelling language. One line of research focuses of the introduction of an appropriate notion of *space*. One possibility is to take StoKLAIM [11] as a starting point, thus using a simple, locality based approach. Another, perhaps more interesting possibility, instead, is to use a richer, predicate based, addressing mechanism, like (a possibly restricted version of) the addressing mechanism of StocS [20], where the location is just one of the agents' attributes and its values are instances of an appropriate data type, namely *space*. This can take different forms, from topological spaces—including bi- or tri-dimensional continuous space—to more general closure spaces—including generic graphs—as in [6]. Another issue is the inclusion of richer data and operations including templates and pattern-matching. This implies the definition of syntactical restrictions, or static analysis techniques, for guaranteeing that in all computations of a model specification the set of *distinct* data values is bounded, while the multiplicity of each item can of course be unbounded. This also holds for the inclusion of process spawning and processes to be stored/retrieved to/from repositories. Finally, we plan to adapt fluid model-checking techniques to tuple-based languages.

References

1. Antonaki, M., Philippou, A.: A process calculus for spatially-explicit ecological models. In: Ciobanu, G. (ed.) Proceedings 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi, MeCBIC 2012, Newcastle, UK, 8th September 2012. EPTCS, vol. 100, pp. 14–28 (2012), http://dx.doi.org/10.4204/EPTCS.100.2
2. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective systems behaviour: A Tutorial. Performance Evaluation - An International Journal. Elsevier 70, 317–349 (2013), doi 10.1016/j.peva.2013.01.001
3. Bortolussi, L., Policriti, A.: Dynamical systems and stochastic programming: To ordinary differential equations and back. T. Comp. Sys. Biology 11, 216–267 (2009), doi:10.1007/978-3-642-04186-0_11
4. Bortolussi, L., Hillston, J.: Fluid model checking. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7454, pp. 333–347. Springer (2012), http://dx.doi.org/10.1007/978-3-642-32940-1_24
5. Carriero, N., Gelernter, D., Mattson, T.G., Sherman, A.H.: The linda® alternative to message-passing systems. Parallel Computing 20(4), 633–655 (1994), http://dx.doi.org/10.1016/0167-8191(94)90032-9
6. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: Diaz, J., Lanese, I., Sangiorgi, D. (eds.) Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8705, pp. 222–235. Springer (2014), http://dx.doi.org/10.1007/978-3-662-44602-7_18
7. Ciocchetta, F., Hillston, J.: Bio-pepa: A framework for the modelling and analysis of biological systems. Theor. Comput. Sci. 410(33-34), 3065–3084 (2009), http://dx.doi.org/10.1016/j.tcs.2009.02.037
8. De Nicola, R., Latella, D., Lluch Lafuente, A., Loreti, M., Margheri, A., Massink, M., Morichetta, A., Pugliese, R., Tiezzi, F., Vandin, A.: The SCEL Language: Design, Implementation, Verification. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems, LNCS, vol. 8998, chap. I.1, pp. 3–71. Springer-Verlag (2015), dOI: 10.1007/978-3-319-16310-9_1, ISBN 978-3-319-16309-3 (print), 978-3-319-16310-9 (online), ISSN 0302-9743
9. De Nicola, R., Latella, D., Loreti, M., Massink, M.: A Uniform Definition of Stochastic Process Calculi. ACM Computing Surveys 46(1), 5:1–5:35 (2013), doi 10.1145/2522968.2522973

10. De Nicola, R., Ferrari, G.L., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. 24(5), 315–330 (1998), http://doi.ieeecomputersociety.org/10.1109/32.685256
11. De Nicola, R., Katoen, J., Latella, D., Loreti, M., Massink, M.: Model checking mobile stochastic logic. Theor. Comput. Sci. 382(1), 42–70 (2007), http://dx.doi.org/10.1016/j.tcs.2007.05.008
12. Deneubourg, J.L., Aron, S., Goss, S., Pasteels, J.M.: The self-organizing exploratory pattern of the argentine ant. Journal of Insects Behaviour 3(2) (1990)
13. Feng, C., Hillston, J.: PALOMA: A Process Algebra for Located Markovian Agents. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 266–280. Springer-Verlag (2014)
14. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the Argentine Ant. Naturwissenschaften 76, 579–581 (1989)
15. Guenther, M.C., Bradley, J.T.: Higher moment analysis of a spatial stochastic process algebra. In: Thomas, N. (ed.) Computer Performance Engineering - 8th European Performance Engineering Workshop, EPEW 2011, Borrowdale, UK, October 12-13, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6977, pp. 87–101. Springer (2011), http://dx.doi.org/10.1007/978-3-642-24749-1_8
16. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing. The International Journal of Formal Methods. Springer-Verlag 6(5), 512–535 (1994)
17. Hermanns, H., Herzog, U., Katoen, J.: Process algebra for performance evaluation. Theor. Comput. Sci. 274(1-2), 43–87 (2002), http://dx.doi.org/10.1016/S0304-3975(00)00305-4
18. Latella, D., Loreti, M., Massink, M.: On-the-fly Fluid Model Checking via Discrete Time Population Models. Extended Version. Technical Report TR-QC-08-2014, QUANTICOL (2014)
19. Latella, D., Loreti, M., Massink, M.: On-the-fly fast mean-field model-checking. In: Abadi, M., Lluch-Lafuente, A. (eds.) Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8358, pp. 297–314. Springer (2013), http://dx.doi.org/10.1007/978-3-319-05119-2_17
20. Latella, D., Loreti, M., Massink, M., Senni, V.: Stochastically timed predicate-based communication primitives for autonomic computing. In: Bertrand, N., Bortolussi, L. (eds.) Proceedings Twelfth International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2014, Grenoble, France, 12-13 April 2014. EPTCS, vol. 154, pp. 1–16 (2014), http://dx.doi.org/10.4204/EPTCS.154.1
21. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications: The TOTA approach. ACM Trans. Softw. Eng. Methodol. 18(4) (2009), http://doi.acm.org/10.1145/1538942.1538945
22. Massink, M., Latella, D.: Fluid analysis of foraging ants. In: Sirjani, M. (ed.) Coordination Models and Languages - 14th International Conference, COORDINATION 2012, Stockholm, Sweden, June 14-15, 2012. Proceedings. LNCS, vol. 7274, pp. 152–165. Springer-Verlag (2012), http://dx.doi.org/10.1007/978-3-642-30829-1_11
23. Tribastone, M., Gilmore, S., Hillston, J.: Scalable differential analysis of process algebra models. IEEE Transactions on Software Engineering. IEEE CS 38(1), 205–219 (2012)