On-the-Fly Mean-Field Model-Checking for Attribute-Based Coordination

Vincenzo Ciancia, Diego Latella, Mieke Massink

On-the-Fly Mean-field Model-checking for Attribute-based Coordination*

Vincenzo Ciancia[1], Diego Latella[1], and Mieke Massink[1]

CNR-ISTI, Italy, {V.Ciancia, D. Latella, M. Massink}@cnr.it

Abstract. Typical Collective Adaptive Systems (CAS) consist of a large number of interacting objects that coordinate their activities in a decentralised and often implicit way. The design of such systems is challenging, as it requires scalable analysis tools and methods to check properties of proposed system designs before they are put into operation. A promising technique is Fast Mean-Field Approximated Model-checking. The FlyFast model-checker uses an on-the-fly algorithm for bounded PCTL model-checking of selected individuals in the context of very large populations whose global behaviour is approximated using deterministic limit techniques. Recently, specific modelling languages have been proposed for CAS. A key feature of such languages is the *attribute-based* interaction paradigm. In this paper we present an attribute-based coordination language as a front-end for FlyFast. Its formal probabilistic semantics is provided and a translation to the original FlyFast language is given and proved correct. Application examples are also provided.

Keywords: Collective Adaptive Systems · Probabilistic On-the-fly Model-Checking · Mean-Field Approximation · Discrete Time Markov Chains

1 Introduction and Related Work

Collective Adaptive Systems (CAS) consist of a large number of entities with decentralised control and varying degrees of complex autonomous behaviour. CAS are at the core of the envisioned smart cities of the future and encompass systems like smart urban transport and smart grids. The pervasive nature of CAS and thus their impact on society requires the development of reliable rigorous design models as well as *a priori* analysis techniques of such models—covering all relevant aspects of their behaviour, including quantitative and emergent ones—before they are put into operation[1].

Model-checking has been widely recognised as a powerful approach to the automatic verification of concurrent and distributed systems. It consists of an efficient procedure that, given an abstract model of the system, decides whether the model satisfies a logical formula, typically drawn from a temporal logic. Unfortunately, traditional model-checking suffers from the so called *state-space*

* Research partially funded by the EU project QUANTICOL (nr. 600708).

[1] See, e.g. www.focas.eu/adaptive-collective-systems and www.quanticol.eu

explosion problem which hampers scalability of the approach. In particular, its application to *very large* models, like those typical of CAS, is infeasible. In [17, 15] Latella et al. presented a scalable mean-field model-checking procedure for verifying bounded PCTL (Probabilistic Computation Tree Logic) [11] properties of selected individuals in the context of systems consisting of a large number of similar, but independent, interacting objects; a limited form of global system properties can be treated as well. The procedure can be used with huge population sizes, as typical of analysis techniques based on mean-field approximation; the average behaviour of the population is approximated using a population Discrete Time Markov Chain (DTMC) convergence result [21] and is used for representing the context in which the selected individuals operate (see [21, 17, 15] for details). The model-checking procedure is implemented in the tool FlyFast as an instantiation of a probabilistic on-the-fly model-checker; the latter is parametric on (the semantic model of) the modelling language [16, 15].

FlyFast comes with simple modelling language. An agent[2] is a finite state process, a generic state C of which is specified by a *state defining equation* like $C := a_1.C_1 + \ldots + a_r.C_r$. Intuitively, the above notation defines state C of the agent and postulates that there are r outgoing transitions from C,with action a_j labelling a transition going from C to C_j. A probability value is assigned to each action a by means of a *probability function definition* $a :: E$, where the actual probability is given by the value of expression E in the current *occupancy measure vector* $\mathbf{m}$. Assume a system is composed of N instances of the agent and that the states of the agent are $C_1, \ldots C_S$. The occupancy measure vector at the current time is the vector $(m_1, \ldots, m_S)$ s.t. m_j yields the *fraction* of agents currently in state C_j over the total number N of agents. A *system specification* is a triple composed by an *agent specification*—given as a set of state defining equations—a set of probability function definitions, and an initial global state. Finally, FlyFast provides the user with *formula declarations* which allow for the interpretation of bounded PCTL atomic propositions in the model at hand. The computational model is *clock-synchronous*; at each step each agent must perform an independent step (which may be an idle self-loop) so that the global state probabilities are given as the product of agent step probabilities, and a new occupancy measure vector can be computed. The global system behaviour is thus a DTMC as well as the stochastic process given by the occupancy measure vector. Notably, for N sufficiently large, the latter can be approximated deterministically, i.e. by a *function* of time. This brings to a dramatic decrease in size of the global state space: at each step, the total number of potential next states drops from S^N to S, which makes bounded PCTL model-checking of very large population systems possible (the interested reader is referred to [17, 15, 21] for details).

Recently, modelling and programming languages have been proposed specifically for autonomic computing systems and CAS [9, 3, 12]. Typically, in such frameworks, a system is composed of a set of independent *components* where a component is a process equipped with a set of *attributes* describing features of the component. A classical example of attribute is the component *location*.

[2] In the context of FlyFast we use the words *agent*, *process* and *object* as synonyms.

An additional *environment* is often used for the specification of common or global features. The attributes of a component can be *updated* during its execution so that the association between attribute *names* and attribute *values* is mantained in the dynamic *store* of the component. Attributes can be used in *predicates* appearing in language constructs for component interaction. For instance a component may *broadcast* a message to *all those components satisfying* a given predicate; similarly a component may wait for a message from *any of those components satisfying* a given predicate.

In the present paper, we propose an extension of the FlyFast front-end modelling language for dealing with *components* and *predicate-based interaction*. The extension has been inspired by CARMA [3]. Components are expressed as pairs *process-store*; actions are *predicate based multi-cast* output and input primitives[3]. Associated to each action there is also an (atomic) probabilistic store-update. For instance, assume components have an attribute named loc which takes values in the set of points of a space type. The following action models a multi-cast via channel α to all components in the same location as the sender, making it change location randomly: $\alpha^*[\mathsf{loc} = \mathbf{my}.\mathsf{loc}]\langle\rangle\{\mathsf{loc} \leftarrow \mathsf{randomLoc}(\mathsf{loc})\}$. Here randomLoc is assumed to be a random generator of points in the space[4]. The computational model is *clock-synchronous* as well, but at the component level. In addition, each component is equipped with a local *outbox*. The effect of an output action $\alpha^*[\pi_r]\langle\rangle\sigma$ is to deliver output label $\alpha\langle\rangle$ to the local outbox, together with the predicate π_r, which receiver components will be required to satisfy, as well as the current store γ of the component executing the action; the current store is updated according to update σ. Note that output actions are *non-blocking* and that successive output actions of the same component rewrite its outbox. An input action $\alpha^*[\pi_s]()\sigma$ by a component will be executed with a probability which is proportional to the *fraction* of all those components whose outboxes currently contain the label $\alpha\langle\rangle$, a predicate π_r which is satisfied by the component, and a store γ which satisfies in turn predicate π_s. If such a fraction is zero, then the input action will not take place (input is blocking), otherwise the action takes place, the store of the component is updated via σ, and its outbox cleared. Thus, as in the original FlyFast language, component interaction is probabilistic, but now the *fraction* of the components satisfying the relevant predicates plays a role in the computation of transition probabilities. We provide the formal probabilistic semantics of the extended language and a translation to the original FlyFast language which makes the model-checker support the extended language. The translation is proved correct.

Related Work. As we mentioned before, this work has been inspired by CARMA [3], which in turn shares features with SCEL [9]. There are several aspects of either languages that are not present in our proposal. The main reason for the absence

[3] For the sake of notational simplicity, in this paper we present a non value-passing version of the FlyFast front-end; the complete, value-passing, approach is described in [8].

[4] *Multi-cast* interaction is denoted using the $\_^*$ notation, as in CARMA.

of most of them is the fact that this work is intended as a proof of concept rather than the realisation of a ready-to-use tool for reasoning about CAS. So we aim at keeping the language minimal and focussing only on attribute-based interaction in the context of stochastic and mean-field semantics and model-checking. A feature of CARMA not considered here is the notion of *global* environment, since it represents a singularity point that does not fit well with limit approximation techniques. Finally, we point out that the stochastic semantics of CARMA are based on *time inhomogeneous* CTMCs, due to the fact that action parameters may be time dependent, while we use DTMCs as semantic basis. The notion of the outbox is reminiscent of the notion of the ether in PALOMA [10] in the sense that the collection of all outboxes together can be though of as a kind of ether; but such a collection is intrinsically distributed among the components so that it cannot represent a bottleneck in the execution of the system neither a singularity point in the deterministic approximation. Fluid model-checking for *continuous time* systems is addressed in [4] where a global model-checking procedure for the Continuous Stochastic Logic (CSL, [2]) is given, which is based on continuous limit approximated semantics. Fluid semantics have proved very useful for reasoning about large coordination systems (see e.g. [23, 6, 18]). Predicate-/attribute-based inter-process communication has been originally proposed in [19] where several variants of predicate-/attribute-based communication primitives—including blocking / non-blocking, bounded / unbounded—are discussed in the context of a study on high-level language constructs for distributed systems with decentralised control (see for instance [22]). The notion of predicate-/attribute-based interaction is central in the definition of SCEL [9] where its synchronous-communication variant has been given formal semantics. Asynchronous-communication variants have been defined for stochastic versions of SCEL [20]. An attribute-interaction based calculus is proposed in [1] where broadcast communication links among components are dynamically established on the basis of the interdependences determined by predicates over attributes. A reduction semantics approach is adopted where each transition involves the group composed of both sender and receivers. Attribute π-Calculus has been proposed in [14] and extended to Imperative π-Calculus in [13]; in both calculi, which inherit the classical point-to-point communication paradigm of the π-Calculus, as opposed to multi-cast, attributes are related to messages rather than to processes. None of the above mentioned works on predicate-/attribute-based languages addresses mean-field approximated model-checking so, to the best of our knowledge, the present paper is the first proposal on the subject.

2 Attribute-based Coordination Language and Logic

In this section we define an attribute-based population description language and related logic. A *system* is defined as a population of N identical interacting components[5] in a *clock-synchronous* fashion. Each component is equipped with

[5] In practice, the fact that the components are identical is not a strong limitation since each component may consist of several different sets of states, with each state

a finite set of *attributes*; the current *store* $\gamma \in \Gamma$ of the component maps each attribute *name* to an attribute *value*.

2.1 Syntax

A *component specification* is a pair (Δ, F) where Δ is a finite set of *state-defining* equations, one for each *state* of the component and F is a set of auxiliary function definitions[6]. We let $\mathcal{S}$, ranged over by $C, C', C_1, \ldots$ denote the (denumerable, non-empty) set of all states which can be used in equations. Each equation defines the transitions from the state to other states of the component; each transition is labelled by the *action* the component performs when the transition takes place. The general format of a state defining equation is: $C := [g_1]P_1 + \ldots + [g_r]P_r$ where each *guard* $[g]$ is a *predicate* π defined according to the following grammar: $\pi ::= \top \mid \bot \mid e_1 \bowtie e_2 \mid \neg\pi \mid \pi_1 \ \wedge \ \pi_2$.
$\top$ ($\bot$, resp.) stands for the truth value *true* (*false* resp.), while $\bowtie \in \{\geq, >, \leq, <\}$; we let $\bowtie \in \{>, <\}$. An *expression* e can be an attribute name a, or **my**.a referring to the value of a in the component where it occurs, or a value v in given set $\mathcal{V}$. In defining equations as above, we abbreviate $[\top]P_j$ with P_j and we omit summands of the form $[\bot]P_j$. Each P_j in a state defining equation as above is of the form $p_j :: act_j.C_j$, where p_j is a probability expression, i.e. an expression with value in $[0, 1]$, built from constants $v \in [0, 1]$ and the special operator $\mathsf{frc}\, C$, combined using standard arithmetic operators; for state C, $\mathsf{frc}\, C$ returns the *fraction* of the components that are currently in state C, over the total of N components. Clearly, the use of the frc operator allows action (and, ultimately, transition) probability to depend on the global state of the system. Actions act_j can be *output* actions $\alpha^*[\pi]\langle\rangle\sigma$ or *input* actions $\alpha^*[\pi]()\sigma$. We assume a countable set of *action types* $\mathcal{A}$, with $\alpha \in \mathcal{A}$. The effect of an *output* action $\alpha^*[\pi]\langle\rangle\sigma$ is a broadcast to all those components satisfying predicate π and which are willing to accept the interaction. This is achieved by means of delivering $\alpha\langle\rangle$, together with some additional information, to the outbox of the component executing the action, as we will discuss in detail in Sect. 2.2. In addition, the store of the component executing the action is updated according to the *update* σ, which is a function from Γ to the class of probability distributions over Γ—i.e., in the general case, the update may be probabilistic. Similarly, an *input* action $\alpha^*[\pi]()\sigma$ is used to receive an α-message sent by a component satisfying predicate π. More specifically, the probability of executing the input action will be proportional to the fraction of components which have sent the α-message while satisfying predicate π and requiring a predicate which is satisfied by the component executing the input action. Also input actions are provided with a store update σ whereas the component outbox is cleared as (a side) effect of their execution. In the sequel, we shall call *address predicates* the predicates $[\pi]$ used for identifying the partners in input/output actions. For updates, we use

in a given set being unreachable from states of other sets. Each such a set of states can be seen as a component with a different behaviour.

[6] The specific syntax of auxiliary function definitions is irrelevant and left out here.

the following notation: $\{a_1 \leftarrow e_1^\gamma, \ldots, a_t \leftarrow e_t^\gamma\}$ where e_j^γ is an expression which may also include functions—the definition of which are to be provided in F—which may depend on the component store γ and produce random results, as we shall see below. Attributes different from $a_1, \ldots, a_t$ are left unchanged by the update. We require that any attribute name a occurring in a guard $[g]$, or in the expressions $e_1^\gamma, \ldots, e_t^\gamma$, must appear in the form **my**.a (thus referring to the value of the attribute in the local store of the component). An attribute name a may appear both with and without the **my**. prefix in the address predicate π. Intuitively, equation $C := [g_1]P_1 + \ldots + [g_r]P_r$ defines state C of the component at hand and postulates that there are r potential outgoing transitions from C, with action act_j labelling a transition going from C to C_j. The actual transitions will be determined by the value of the guards and the action probabilities. Note that it may happen that the current cumulative probability value of the enabled transitions is less than 1; for this reason, the language provides the construct **rest** :: $\alpha[\pi]\langle\rangle\sigma.C$, where **rest** is defined as the residual probability; it is required that there is at most one **rest**-branch (typically the last one) in every state defining equation. Only output actions are allowed in **rest**-branches; this ensures that the residual probability is not affected by the fraction of those components in the system satisfying the address predicate. Obviously, in a given component specification there is exactly one defining equation for each state of the component. We let $\mathcal{S}_\Delta$ denote the finite set of states defined by Δ. Similarly, Γ_Δ, $\mathcal{A}_\Delta$ and Π_Δ denote the set of all stores associated to Δ, the action types and the predicates occurring in (the equations of) Δ. Finally, we let $\mathcal{V}_\Delta$ denote the set of values which can be taken by the attributes of a component specified by Δ. Note that we assume $\mathcal{V}_\Delta$ is a finite set—thus also Γ_Δ is finite; model finiteness is a common assumption for modelling languages supported by automatic analysis and verification tools.

Example 1 (A spatially distributed Computer Epidemic Model). We enrich the Computer Epidemic Model of [5], SEIR, with infection communication and a *bi-dimen-sional Regular GRID* [7] model for space, where for each point ℓ the following specific operators are defined, with the usual North, South, East, West meaning: $\mathsf{N}(\ell), \mathsf{S}(\ell), \mathsf{E}(\ell), \mathsf{W}(\ell)$. Each component is equipped with a *position* attribute, named loc, which is always yielding the current position (i.e. point) in space of the component and is the only attribute of the component. Note that, given the abstract nature of the bi-dimensional Regular GRID, such a "point" could be a physical point is space, but also a specific region (or patch) in a patched representation of space. We will implicitly refer to the second interpretation in the sequel. In the model, given in Fig. 1, the purpose of auxiliary function `Jump` is twofold: (i) it defines a function from positions to discrete probability distributions which, given position ℓ, characterizes a probability distribution which assignes probability $p_{\mathsf{N}}(\ell)$ to $\mathsf{N}(\ell)$, probability $p_{\mathsf{S}}(\ell)$ to $\mathsf{S}(\ell)$, and so on and (ii) defines a random position generator which, given position ℓ, randomly returns a new position according to the specified probabilities. Note that the probabilities are themselves functions of the position and they are assumed being declared as additional auxiliary functions. In the equation for `S` in Fig. 1,

$$\mathtt{Jump}(\ell) := [p_{\mathsf{H}}(\ell) :: \ell;\ p_{\mathsf{N}}(\ell) :: \mathsf{N}(\ell);\ p_{\mathsf{S}}(\ell) :: \mathsf{S}(\ell);\ p_{\mathsf{E}}(\ell) :: \mathsf{E}(\ell);\ p_{\mathsf{W}}(\ell) :: \mathsf{W}(\ell)]$$

$$\begin{aligned}
\mathtt{S} := {} & h :: \mathtt{inf}^*[\mathsf{loc} = \mathbf{my}.\mathsf{loc}]()\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E} + \\
& m_N :: \mathtt{inf}^*[\mathsf{loc} = \mathsf{N}(\mathbf{my}.\mathsf{loc})]()\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E} + \\
& m_S :: \mathtt{inf}^*[\mathsf{loc} = \mathsf{S}(\mathbf{my}.\mathsf{loc})]()\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E} + \\
& m_E :: \mathtt{inf}^*[\mathsf{loc} = \mathsf{E}(\mathbf{my}.\mathsf{loc})]()\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E} + \\
& m_W :: \mathtt{inf}^*[\mathsf{loc} = \mathsf{W}(\mathbf{my}.\mathsf{loc})]()\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E} + \\
& ext :: \mathtt{ext}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E} + \\
& sr :: \mathtt{rec}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{R} + \\
& \mathbf{rest} :: \mathtt{nsc}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{S}
\end{aligned}$$

$$\begin{aligned}
\mathtt{E} := {} & ei :: \mathtt{act}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{I} + \\
& er :: \mathtt{rec}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{R} + \\
& \mathbf{rest} :: \mathtt{nsc}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{E}
\end{aligned}$$

$$\begin{aligned}
\mathtt{I} := {} & ii :: \mathtt{inf}^*[\top]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{I} + \\
& ir :: \mathtt{rec}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{R} + \\
& \mathbf{rest} :: \mathtt{nsc}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{I}
\end{aligned}$$

$$\begin{aligned}
\mathtt{R} := {} & rs :: \mathtt{loss}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{S} \\
& \mathbf{rest} :: \mathtt{nsc}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}.\mathtt{R}
\end{aligned}$$

Fig. 1. A four state model: *susceptible* (S), *exposed* (E), *infected* (I), and *recover* (R).

probability constants $h, m_N, \ldots, m_W$ are factors in $[0, 1]$ with cumulative value at most 1, each to be multiplied by the actual probability of the associated (input) action. The latter will be computed as the fraction of the local states which satisfy the required predicate. The resulting values, when taken all together, will characterize a probability *sub*-distribution; the residual probability will be associated to a **rest**-self-loop. Similar considerations apply to the probability constants in the definition of other states (e.g. i in the figure). We assume $h > m_N \approx m_S \approx m_E \approx m_W$. In other words, an agent has higher probability to get the infection from agents in the same place than from agents in adjacent places; the probability drops to zero in all other cases. •

A system is modelled as a population of N instances of a component, so a *system specification* Υ is a triple $(\Delta, F, \boldsymbol{\Sigma_0})^{(N)}$ where (Δ, F) is a component specification and $\boldsymbol{\Sigma_0}$ is the *initial (system) global state*, which will be discussed below. In the sequel we will often write Δ instead of (Δ, F).

2.2 Probabilistic Semantics

In order to model component interactions within a system, each component is equipped with a local outbox. The idea is that, whenever a component executes an output action, the related output will be available in the component's outbox *only during the next clock tick*; in the next state, (other) components

will be able to get the message by means of corresponding input actions. After such a tick, the outbox will be empty or filled with the information generated by a subsequent output action of the component. Formally, let Λ^O_Δ be the set $\Lambda^O_\Delta = \{\alpha\langle\rangle | \alpha \in \mathcal{A}_\Delta\}$. An *outbox-state* $O \in \mathcal{O}_\Delta = \{\langle\rangle\} \cup (\Gamma_\Delta \times \Pi_\Delta \times \Lambda^O_\Delta)$ is either empty or a triple $(\gamma, \pi, \alpha\langle\rangle)$. A *component-state* Σ is a triple $\Sigma = (C, \gamma, O) \in \mathcal{S}_\Delta \times \Gamma_\Delta \times \mathcal{O}_\Delta = \Omega_\Delta$, where C, γ, O are the current state, store and outbox-state of the component, respectively. If the component-state is the target of a transition modelling the execution of an *output* action, then $O = (\gamma', \pi, \alpha\langle\rangle)$, where γ' is the store of the (component-state) source of the transition, π is the predicate used in the action—actualized with γ'—and $\alpha\langle\rangle$ the actual message sent by the action. If, instead, the component-state is the target of a transition for an *input* action, then $O = \langle\rangle$, i.e. the empty outbox. A *global state* is a tuple $\boldsymbol{\Sigma} = ((C_1, \gamma_1, O_1), \ldots, (C_N, \gamma_N, O_N)) \in \Omega^N_\Delta$ where $\boldsymbol{\Sigma}_{[j]} = (C_j, \gamma_j, O_j)$ is the component-state of the j-th instance in the population for $j = 1 \ldots N$. We say that N is the *population size* of the system. In the sequel, we will omit the explicit indication of the size N in $(\Delta, F, \boldsymbol{\Sigma_0})^{(N)}$, and elements thereof or related functions, writing simply $(\Delta, F, \boldsymbol{\Sigma_0})$, when this cannot cause confusion. In summary, a system specification can be thought of as process algebraic clock-synchronous parallel composition of N processes. The probabilistic behaviour of a system can be derived from its specification $(\Delta, F, \boldsymbol{\Sigma_0})^{(N)}$. We remind that Ω_Δ is finite, since so are sets $\mathcal{S}_\Delta, \Gamma_\Delta$ and $\mathcal{O}_\Delta$. Assume $\Omega_\Delta = \{\Sigma_1, \ldots, \Sigma_S\}$ and let $\mathcal{U}^S$ be the set $\{\mathbf{m} \in [0,1]^S | \sum_{i=1}^S \mathbf{m}_{[i]} = 1\}$; we can assume, w.l.o.g. that there is a total ordering on Ω_Δ so that we can unambiguously associate each component m_j of a vector $\mathbf{m} = (m_1, \ldots, m_S) \in \mathcal{U}^S$ with a distinct element Σ_j of $\{\Sigma_1, \ldots, \Sigma_S\}$. With each global state $\boldsymbol{\Sigma}^{(N)}$ an *occupancy measure* vector $\mathbf{M}^{(N)}(\boldsymbol{\Sigma}^{(N)}) \in \mathcal{U}^S$ is associated where $\mathbf{M}^{(N)}(\boldsymbol{\Sigma}^{(N)}) = (M_1^{(N)}, \ldots, M_S^{(N)})$ with $M_i^{(N)} = \frac{1}{N} \sum_{n=1}^N \mathbf{1}_{\{\boldsymbol{\Sigma}^{(N)}_{[n]} = \Sigma_i\}}$ for $i = 1, \ldots, S$ and the value of $\mathbf{1}_{\{\alpha=\beta\}}$ is 1, if $\alpha = \beta$, and 0 otherwise. So, for $\Sigma_i = (C_i, \gamma_i, O_i)$, $M_i^{(N)}$ is the *fraction*, in the current global state $\boldsymbol{\Sigma}^{(N)}$, of the component instances which are in state C_i, have store γ_i and outbox O_i, over the total number N. We assume semantic interpretation functions $\mathbf{E_L}[\![\cdot]\!]$ and $\mathbf{E_R}[\![\cdot]\!]$ for the *local*, *remote* respectively, interpretation of expressions and predicates and a function $\mathbf{E_P}[\![\cdot]\!]$ for the interpretation of probability expressions. $\mathbf{E_L}[\![e]\!]$ ($\mathbf{E_R}[\![e]\!]$, respectively) takes a local (remote, respectively) store γ as an argument, whereas $\mathbf{E_P}[\![p]\!]$ takes an occupancy measure vector $\mathbf{m}$ as an argument. We note that $\mathbf{E_L}[\![a]\!]_\gamma = a, \mathbf{E_L}[\![\mathbf{my}.a]\!]_\gamma = \gamma(a)$, $\mathbf{E_R}[\![a]\!]_\gamma = \gamma(a)$, and $\mathbf{E_P}[\![\mathsf{frc}\, C]\!]_\mathbf{m} = \sum_{i=1}^S \{\mathbf{m}_{[i]} | \Sigma_i = (C, \gamma_i, O_i)\}$; moreover, $\mathbf{E_R}[\![\mathbf{my}.a]\!]_\gamma$, $\mathbf{E_P}[\![\mathrm{tt}]\!]_\mathbf{m}$, $\mathbf{E_P}[\![\mathrm{ff}]\!]_\mathbf{m}$, $\mathbf{E_L}[\![\mathsf{frc}\, C]\!]_\gamma$, and $\mathbf{E_R}[\![\mathsf{frc}\, C]\!]_\gamma$ are undefined as are, for the sake of simplicity, $\mathbf{E_P}[\![a]\!]_\mathbf{m}$, $\mathbf{E_P}[\![\mathbf{my}.a]\!]_\mathbf{m}$. The definition of the above semantic interpretation functions on composition terms can be given recursively on the structure of the terms and is left out here. In particular, we assume them extended to tuples. Similarly, we assume standard techniques and machinery for auxiliary functions

in store updates; the semantics of update σ in the current store γ will be denoted by $\mathbf{E_U}[\![\sigma]\!]_\gamma$, that is a probability distribution over stores[7].

Let Λ_Δ be defined as $\Lambda_\Delta = \Lambda^O_\Delta \cup \Lambda^I_\Delta$, with Λ^O_Δ as above, and $\Lambda^I_\Delta = \{\alpha()|\alpha \in \mathcal{A}_\Delta\}$. A component specification (Δ, F) characterises the (component) transition probability matrix as a function of occupancy measure vectors $\mathbf{m}$, $\mathbf{K}^{(N)} : \mathcal{U}^S \times \Omega_\Delta \times \Omega_\Delta \to [0,1]$ such that $\mathbf{K}^{(N)}(\mathbf{m})_{\Sigma,\Sigma'}$ is the probability of a one step jump from component-state Σ to component-state Σ', given (that the global system state induces) occupancy measure vector $\mathbf{m}$. $\mathbf{K}^{(N)}(\mathbf{m})_{\Sigma,\Sigma'}$ is computed by making use of a transition relation $(C,\gamma,O) \xrightarrow{\lambda,p} (C',\gamma',O')$ over the space of component-states Ω_Δ, with transition labels drawn from $\Theta_\Delta \subset \Lambda_\Delta \times [0,1]$. More specifically, the transition relation is the relation $\to \subseteq \Omega_\Delta \times \Theta_\Delta \times \Omega_\Delta$ such that $(C,\gamma,O) \xrightarrow{\lambda,p} (C',\gamma',O')$ iff $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ is the defining equation for C and $p = \sum_{k\in J}\{\bar{p}_k | (C,\gamma,O) \xrightarrow{\lambda,\bar{p}_k}_k (C',\gamma',O')\}$, where $\xrightarrow{\lambda,\bar{p}_k}_k$ is the least relation induced by the rules in Fig. 2. The component transition matrix function

$$\frac{C := \sum_{j\in J}[g_j]p_j :: act_j.C_j \quad k \in J \quad \mathbf{E_L}[\![g_k]\!]_\gamma = \mathrm{tt} \quad act_k = \alpha^*[\pi]\langle\rangle\sigma \quad p = \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma')}{(C,\gamma,O) \xrightarrow{\alpha\langle\rangle, p\cdot \mathbf{E_P}[\![p_k]\!]_\mathbf{m}}_k (C_k, \gamma', (\gamma, \mathbf{E_L}[\![\pi]\!]_\gamma, \alpha\langle\rangle))} \tag{1}$$

$$\frac{C := \sum_{j\in J}[g_j]p_j :: act_j.C_j \quad k \in J \quad [g_k]p_k = \mathbf{rest} \quad act_k = \alpha^*[\pi]\langle\rangle\sigma \quad p = \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma')}{(C,\gamma,O) \xrightarrow{\alpha\langle\rangle, p\cdot(1-\sum_{j\in(J\setminus\{k\})}\mathbf{E_P}[\![p_j]\!]_\mathbf{m})}_k (C_k, \gamma', (\gamma, \mathbf{E_L}[\![\pi]\!]_\gamma, \alpha\langle\rangle))} \tag{2}$$

$$\frac{\begin{array}{c} C := \sum_{j\in J}[g_j]p_j :: act_j.C_j \quad k \in J \quad \mathbf{E_L}[\![g_k]\!] = \mathrm{tt} \\ act_k = \alpha^*[\pi]()\sigma \quad p = \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') \\ f = \sum_{i=1}^S \{\mathbf{m}_{[i]} | \Sigma_i = (C'_i, \gamma'_i, (\gamma''_i, \pi'_i, \alpha\langle\rangle)) \ \wedge \\ \mathbf{E_R}[\![\pi'_i]\!]_\gamma = \mathbf{E_R}[\![\mathbf{E_L}[\![\pi]\!]_\gamma]\!]_{\gamma''_i} = \mathrm{tt}\} \end{array}}{(C,\gamma,O) \xrightarrow{\alpha(), p\cdot\mathbf{E_P}[\![p_k]\!]_\mathbf{m}\cdot f}_k (C_k, \gamma', \langle\rangle)} \tag{3}$$

Fig. 2. Probabilistic Semantics Rules

$\mathbf{K}^{(N)}(\mathbf{m})_{\Sigma,\Sigma'}$ is defined as follows: $\mathbf{K}^{(N)}(\mathbf{m})_{\Sigma,\Sigma'} = \sum_{(\lambda,p)\in\Theta_\Delta}\{p | \Sigma \xrightarrow{\lambda,p} \Sigma'\}$. Note that all the above summations are finite under our assumption that so is $\mathcal{V}_\Delta$. The behaviour of the system is the result of the parallel-synchronous execution of the N instances of the component. Thus, the probabilistic behaviour of the system is characterised by the DTMC $\mathbf{X}^{(N)}(t)$ with initial probability distribution $\boldsymbol{\delta}_{\boldsymbol{\Sigma_0}}$ and one step probability matrix $\mathbf{P}^{(N)}$ defined by the following product: $\mathbf{P}^{(N)}_{\boldsymbol{\Sigma},\boldsymbol{\Sigma'}} = \Pi_{n=1}^N \mathbf{K}^{(N)}(\mathbf{M}^{(N)}(\boldsymbol{\Sigma}))_{\boldsymbol{\Sigma}_{[n]},\boldsymbol{\Sigma'}_{[n]}}$. Of course, the 'occupancy measure' view of the evolution in time of stochastic process $\mathbf{X}^{(N)}(t)$ is again a DTMC, namely the *occupancy measure DTMC*, which is defined as expected: $\mathbf{M}^{(N)}(t) = \mathbf{M}^{(N)}(\mathbf{X}^{(N)}(t))$.

[7] In this paper, for the sake of simplicity, updates do not depend on the current occupancy measure vector, i.e. the frc operator cannot occur in their specification.

2.3 Bounded PCTL

We recall that, given a set $\mathscr{P}$ of atomic propositions, the syntax of PCTL *state formulas* Φ and *path formulas* φ is defined as follows, where $\mathsf{ap} \in \mathscr{P}$ and $k \geq 0$: $\Phi ::= \mathsf{ap} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\bowtie p}(\varphi)$ where $\varphi ::= \mathcal{X}\Phi \mid \Phi\mathcal{U}^{\leq k}\Phi$. PCTL formulas are interpreted over *state labelled* DTMCs, which are pairs $(\mathcal{M}, L)$ where $\mathcal{M}$ is a DTMC and L is a mapping from the set of states of $\mathcal{M}$ to $2^{\mathscr{P}}$; for each state s, $L(s)$ is the set of atomic propositions true in s[8]. For the purposes of FlyFast bounded PCTL model-checking, our system specifications are enriched with the declaration of three different kinds of atomic propositions. A declaration of the form $\mathsf{ap}\,\mathsf{at}\,C$ associates atomic proposition ap to state $C \in \mathcal{S}_\Delta$. Thus ap must be included in the set $L(\boldsymbol{\Sigma})$ for each global state $\boldsymbol{\Sigma} = ((C_1, \gamma_1, O_1), \ldots, (C_N, \gamma_N, O_N))$ such that $C_1 = C$ (recall here that FlyFast performs model-checking of *the first* object in the context of the global system). A declaration of the form $\mathsf{ap}\,\mathsf{def}\,(\mathbf{my}.a \bowtie v)$ associates atomic proposition ap to all component-states (C, γ, O) s.t. attribute a is $\bowtie v$. So, ap must be included in the set $L(\boldsymbol{\Sigma})$ for each global state $\boldsymbol{\Sigma} = ((C_1, \gamma_1, O_1), \ldots, (C_N, \gamma_N, O_N))$ such that $\mathbf{E}_{\mathbf{L}}[\![\mathbf{my}.a \bowtie v]\!]_{\gamma_1} = \mathrm{tt}$. Finally, a limited form of *global* atomic predicate is provided by means of a declaration of the form $\mathsf{ap}\,\mathsf{def}\,(\mathsf{frc}\,C \bowtie v)$; in this case, ap must be included in the set $L(\boldsymbol{\Sigma})$ for each global state $\boldsymbol{\Sigma}$ s.t. the fraction in $\boldsymbol{\Sigma}$ of the component states (C, γ, O), for any γ and O, is $\bowtie$ than $v \in [0, 1]$.

3 A Translation to FlyFast

In this section we define a translation $\mathcal{I}$ such that, given system specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \boldsymbol{\Sigma_0})^{(N)}$, $\mathcal{I}(\Upsilon) = \langle \Delta, A, \mathbf{C}_0 \rangle^{(N)}$ is a FlyFast [15, 17] system specification preserving probabilistic semantics. The attribute-based FlyFast front-end is then completed with a simple translation at the PCTL level, also provided in this section. We map every component state of Υ to a distinct state of $\mathcal{I}(\Upsilon)$ by means of a total injection $\mathcal{I}_\mathcal{S} : \Omega_{\Delta_\Upsilon} \to \mathcal{S}$. The mapping of actions is a bit more delicate because we have to respect FlyFast static constraints and, in particular, we have to avoid multiple probability function definitions for the same action. To that purpose, we could distinguish different occurrences of the same action in different transitions, characterized by their source and target states in Ω_{Δ_Υ}. In practice, since an action of a component cannot be influenced by the current outbox of the component, it is sufficient to use a total injection $\mathcal{I}_\mathcal{A}$ of the following type $(\mathcal{S}_{\Delta_\Upsilon} \times \Gamma_{\Delta_\Upsilon}) \times \Lambda_{\Delta_\Upsilon} \times \Omega_{\Delta_\Upsilon} \to \mathcal{A}$ for the mapping of actions. In the sequel we show how to build $\mathcal{I}(\Upsilon) = \langle \Delta, A, \mathbf{C}_0 \rangle^{(N)}$ from $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \boldsymbol{\Sigma_0})^{(N)}$. The translation algorithm is given in Fig. 3, where for action $act \in \{\alpha^*[\pi]\langle\rangle\sigma, \alpha^*[\pi]()\sigma\}$ we let $T(act) = \alpha$, $P(act) = \pi$, and $U(act) = \sigma$. $\mathsf{SUM}\{t|\mathsf{cond}(t)\}$ denotes the *syntactical* term representing the sum of terms $t \in \{t|\mathsf{cond}(t) = \mathrm{tt}\}$, i.e. $t_1 + \ldots + t_n$, if $\{t|\mathsf{cond}(t) = \mathrm{tt}\} = \{t_1, \ldots, t_n\} \neq \emptyset$ and 0 if $\{t|\mathsf{cond}(t) = \mathrm{tt}\}$ is the empty set. Finally, by $t * t'$ we mean the *syntactical* term representing the product of terms

[8] We refer to [11] for the formal definition of PCTL and to [17, 15] for the details of its instantiation in FlyFast.

t and t'. Output actions are dealt with in step 1. Consider for example action

For each state equation $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ in Δ_Υ:

1. For each *output* action $\alpha^*[\pi]\langle\rangle\sigma = act_k$ with $k \in J$, and $\gamma \in \Gamma_{\Delta_\Upsilon}$, let $J_{\alpha,\sigma,\gamma}$ be the largest subset of J s.t. there is $C' \in \mathcal{S}_\Delta$ s.t. for all $j \in J_{\alpha,\sigma,\gamma}$ $C_j = C'$, $T(act_j) = \alpha$, $\mathbf{E_L}[\![P(act_j)]\!]_\gamma = \mathbf{E_L}[\![\pi]\!]_\gamma$, and $\mathbf{E_U}[\![U(act_j)]\!]_\gamma = \mathbf{E_U}[\![\sigma]\!]_\gamma$. For each $\gamma' \in \Gamma_{\Delta_\Upsilon}$, with $\xi = \mathcal{I}_\mathcal{A}((C,\gamma),\alpha\langle\rangle,(C',\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$, the following action probability function definition is included in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathsf{SUM}\{p_j | j \in \hat{J}_{\alpha,\sigma,\gamma}\}$, where $\hat{J}_{\alpha,\sigma,\gamma} = \{j \in J_{\alpha,\sigma,\gamma} | [g_j]p_j \neq \mathbf{rest} \ \wedge \ \mathbf{E_L}[\![g_j]\!]_\gamma = \mathrm{tt}\}$. Moreover, for each outbox $O \in \mathcal{O}_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathcal{I}_\mathcal{S}(C,\gamma,O)$: $\xi.\mathcal{I}_\mathcal{S}(C',\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle))$;
2. For each *input* action $\alpha^*[\pi]()\sigma = act_k$, with $k \in J$, and $\gamma \in \Gamma_{\Delta_\Upsilon}$, let $J_{\alpha,\sigma,\gamma}$ be the largest subset of J s.t. there is $C' \in \mathcal{S}_\Delta$ s.t. for all $j \in J_{\alpha,\sigma,\gamma}$ $C_j = C'$, $T(act_j) = \alpha$ and $\mathbf{E_U}[\![U(act_j)]\!]_\gamma = \mathbf{E_U}[\![\sigma]\!]_\gamma$. For each $\gamma' \in \Gamma_{\Delta_\Upsilon}$, with $\xi = \mathcal{I}_\mathcal{A}((C,\gamma),\alpha(),(C',\gamma',\langle\rangle))$, the following action probability function definition is included in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathsf{SUM}\{(p_j * \Phi_j) | j \in \hat{J}_{\alpha,\sigma,\gamma}\}$, where $\hat{J}_{\alpha,\sigma,\gamma} = \{j \in J_{\alpha,\sigma,\gamma} | [g_j]p_j \neq \mathbf{rest} \ \wedge \ \mathbf{E_L}[\![g_j]\!]_\gamma = \mathrm{tt}\}$ and term Φ_j is: $\mathsf{SUM}\{\mathsf{frc}\,\mathcal{I}_\mathcal{S}((\tilde{C},\tilde{\gamma},(\bar{\gamma},\bar{\pi},\alpha\langle\rangle))) | \mathbf{E_R}[\![\bar{\pi}]\!]_\gamma = \mathbf{E_R}[\![\mathbf{E_L}[\![\pi_j]\!]_\gamma]\!]_{\bar{\gamma}} = \mathrm{tt}\}$. Moreover, for each outbox $O \in \mathcal{O}_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathcal{I}_\mathcal{S}(C,\gamma,O)$: $\xi.\mathcal{I}_\mathcal{S}(C',\gamma',\langle\rangle)$;
3. If there exists $k \in J$ s.t. $[g_k]p_k = \mathbf{rest}$, and $act_k = \alpha^*[\pi]\langle\rangle\sigma$, let $\bar{A}$ be the set of probability function definitions which has been constructed in steps (1) and (2) above; note that for every $\zeta :: p * q \in \bar{A}$, q is either a probability constant p_j occurring in a branch $[g_j]p_j :: act_j.C_j$ of the defining equation for C, or it is a term of the form $\mathsf{SUM}\{(p_h * \Phi_h) | h \in H\}$, for some index set H. We define $\bar{p} = (1 - \mathsf{SUM}\{q | \zeta :: p * q \in \bar{A}\})$ and, for all $\gamma' \in \Gamma_{\Delta_\Upsilon}$, with $\xi = \mathcal{I}_\mathcal{A}((C,\gamma),\alpha\langle\rangle,(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$, the following action probability function definition is included in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \bar{p}$. Moreover, for each outbox $O \in \mathcal{O}_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathcal{I}_\mathcal{S}(C,\gamma,O)$: $\xi.\mathcal{I}_\mathcal{S}(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle))$;
4. No other action probability function definition and transition is included and the initial state $\mathbf{C}_0$ of $\mathcal{I}(\Upsilon)$ is defined as $\mathbf{C}_0 = \mathcal{I}_\mathcal{S}(\boldsymbol{\Sigma_0})$.

Fig. 3. The translation algorithm

$\mathtt{ext}^*[\bot]\langle\rangle\{\mathbf{my}.\mathsf{loc} \leftarrow \mathtt{Jump}(\mathbf{my}.\mathsf{loc})\}$ in the definition of state $\mathtt{S}$ in Fig. 1. Suppose the possible values for locations are A, B, C, D, so that stores are functions in $\{\mathsf{loc}\} \to \{A, B, C, D\}$. The algorithm generates 12 actions (diagonal jumps are not contemplated in the example). Let us focus on the action ξ associated to local position A (i.e. $\gamma = [\mathsf{loc} \mapsto A]$) and possible next position B (i.e. $\gamma' = [\mathsf{loc} \mapsto B]$); the algorithm will generate probability function definition $\xi :: \mathtt{p}_\mathsf{W}(A) * ext$ as well as a transition leading to (a state which is the encoding, via $\mathcal{I}_\mathcal{S}$, of) the component state with E as (proper) state, store γ', and outbox $(\gamma, \bot, \mathtt{ext}\langle\rangle)$. Since the action is not depending on the current outbox, in practice a copy of

such a transition is generated *for each* component state sharing the same proper state S and the same store γ. In the general case, in a defining equation for a state C there might be multiple occurrences of the same action, bringing to the same next state C'; the algorithm takes care of this and collects them in order to generate a single transition; the appropriate probability is expressed by means of the $\mathsf{SUM}\{\ldots\}$ term. The translation scheme for input actions is defined in case 2 and is similar, except that for each term p_j in the $\mathsf{SUM}\{\ldots\}$ expression one has also to consider the sum Φ_j of the fractions of the possible partners. The translation of the **rest** case is straighforward.

Let $\mathbf{K}^{(N)}_{\mathcal{I}(\Upsilon)} : \mathcal{U}^S \times \mathcal{I}_{\mathcal{S}}(\Omega_\Delta) \times \mathcal{I}_{\mathcal{S}}(\Omega_\Delta) \to [0,1]$ be the step probability function associated to $\mathcal{I}(\Upsilon)$ by the FlyFast language probabilistic semantics definition (see [15, 17] for details) and $\mathbf{K}^{(N)}_{\Upsilon} : \mathcal{U}^S \times \Omega_\Delta \times \Omega_\Delta \to [0,1]$ be the step probability function for Υ as defined in Sect. 2.2. It is easy to see that:

Theorem 1. *For all $N > 0$, occupancy measure vector $\mathbf{m} \in \mathcal{U}^S$ and $\Sigma, \Sigma' \in \Omega_\Delta$ the following holds: $\mathbf{K}^{(N)}_{\Upsilon}(\mathbf{m})_{\Sigma,\Sigma'} = \mathbf{K}^{(N)}_{\mathcal{I}(\Upsilon)}(\mathbf{m})_{\mathcal{I}_{\mathcal{S}}(\Sigma),\mathcal{I}_{\mathcal{S}}(\Sigma')}$.*

Proof (scketch). We first observe that, by definition, $\mathbf{K}^{(N)}_{\Upsilon}(\mathbf{m})_{(C,\gamma,O),(C',\gamma',O')} = \sum_{(\lambda,p)\in\Theta_\Delta}\{p|(C,\gamma,O) \xrightarrow{\lambda,p} (C',\gamma',O')\}$ which, by definition of $\longrightarrow$, is equal to

$$\sum_{(\lambda,p)\in\Theta_\Delta}\{p|p = \sum_{k\in J}\{\bar{p}_k|(C,\gamma,O) \xrightarrow{\lambda,\bar{p}_k}_k (C',\gamma',O')\} \ \wedge \\ C := \sum_{j\in J}[g_j]p_j :: act_j.C_j \text{ is the def. eq. of } C \text{ in } \Upsilon\}.$$

Consider the outer summation and suppose $(\alpha\langle\rangle, p)$ be the index of a summand. Without loss of generality, assume there is only one instance of such a summand and there is only one $k \in J$ such that the following transition is derived using the rules of Fig. 2: $(C,\gamma,O) \xrightarrow{\alpha\langle\rangle,\bar{p}_k}_k (C',\gamma',O')$. So, we have $\mathbf{K}^{(N)}_{\Upsilon}(\mathbf{m})_{(C,\gamma,O),(C',\gamma',O')} = \bar{p}_k$ such that $(C,\gamma,O) \xrightarrow{\alpha\langle\rangle,\bar{p}_k}_k (C',\gamma',O')$, where $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ is the defining equation for C. Suppose $[g_k]p_k \neq$ **rest**, so that Rule (1) of Fig. 2 has been used for generating the transition. This implies that $\mathbf{E_L}[\![g_k]\!]_\gamma = \mathrm{tt}$, $\bar{p}_k = \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') \cdot \mathbf{E_P}[\![p_k]\!]_{\mathbf{m}}$, $C' = C_k$, and $O' = (\gamma, \mathbf{E_L}[\![\pi]\!]_\gamma, \alpha\langle\rangle)$. Under the above conditions, by definition of the translation algorithm, the action $\xi = \mathcal{I}_{\mathcal{A}}((C,\gamma),\alpha\langle\rangle,(C',\gamma',O'))$ and related action probability function definition $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathsf{SUM}\{p_k\}$ are included in the FlyFast model. Moreover, the summand $\xi.\mathcal{I}_{\mathcal{S}}(C',\gamma',O')$ is added in the equation for state $\mathcal{I}_{\mathcal{S}}(C,\gamma,O)$ in the FlyFast model. Using the semantics definition of the FlyFast language [15, 17], we get that the probability assigned to ξ is $\mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') \cdot \mathbf{E_P}[\![p_k]\!]_{\mathbf{m}}$, that is, exactly $\bar{p}_k$. Thus $\mathbf{K}^{(N)}_{\Upsilon}(\mathbf{m})_{\Sigma,\Sigma'} = \mathbf{K}^{(N)}_{\mathcal{I}(\Upsilon)}(\mathbf{m})_{\mathcal{I}_{\mathcal{S}}(\Sigma),\mathcal{I}_{\mathcal{S}}(\Sigma')}$. The proof for all the other cases is similar. •

The translation of atomic proposition declarations into FlyFast formula declarations is the obvious one and is shown in Fig. 4 where $\mathsf{OR}\{e|\mathsf{cond}(e)\}$ denotes the *syntactical* term representing the disjunction of expressions $e \in \{e|\mathsf{cond}(e) = \mathrm{tt}\}$, i.e. $e_1|\ldots|e_n$, if $\{e|\mathsf{cond}(e) = \mathrm{tt}\} = \{e_1,\ldots,e_n\} \neq \emptyset$ and ff, if $\{e|\mathsf{cond}(e) = \mathrm{tt}\}$ is the empty set.

atomic prop. decl.	FlyFast formula declaration
$\mathsf{ap}\,\mathsf{at}\,C$	$\mathsf{ap} : \mathsf{OR}\{\mathcal{I}_{\mathcal{S}}((C',\gamma,O))\mid(C',\gamma,O)\in\Omega_{\Delta_\Upsilon}\ \wedge\ C'=C\}$
$\mathsf{ap}\,\mathsf{def}\,(\mathbf{my}.a\bowtie v)$	$\mathsf{ap} : \mathsf{OR}\{\mathcal{I}_{\mathcal{S}}((C,\gamma,O))\mid(C,\gamma,O)\in\Omega_{\Delta_\Upsilon}\ \wedge\ \gamma(a)\bowtie v\}$
$\mathsf{ap}\,\mathsf{def}\,(\mathsf{frc}\,C\ \bowtie\ v)$	$\mathsf{ap} : \mathsf{SUM}\{\mathsf{frc}\,\mathcal{I}_{\mathcal{S}}((C',\gamma,O))\mid(C',\gamma,O)\in\Omega_{\Delta_\Upsilon}\ \wedge\ C'=C\}\ \bowtie\ v$

Fig. 4. Translation of atomic proposition declarations. The translation is not defined whenever $\mathsf{OR}\{t|\mathsf{cond}(t)\}=\mathrm{ff}$ or $\mathsf{SUM}\{t|\mathsf{cond}(t)\}=0$.

4 Epidemic Example Revisited

We return to the distributed Epidemic example of Fig. 1 where, for the sake of simplicity, we consider a simple patched space, consisting of the usual four quadrants A, B, C, D in the Cartesian Plane, as in Fig. 5 (left). We model a 'flow' from quadrant C to quadrant A by defining the jump probabilities as in the table in Fig. 5 (right)[9], where $l = 0.6$ and $s = 1 - l$, so that $l > s$.

B	A
C	D

	$\mathtt{p}_H$	$\mathtt{p}_N$	$\mathtt{p}_S$	$\mathtt{p}_E$	$\mathtt{p}_W$
A	l	0	$s/2$	0	$s/2$
B	$s/2$	0	$s/2$	l	0
C	s	$l/2$	0	$l/2$	0
D	$s/2$	l	0	0	$s/2$

Fig. 5. The Cartesian quadrants (left) and the jump probabilities (right).

We consider a model in which initially there are 10.000 components in state S in quadrant C and 100 in state S in quadrant A. The non-zero values of the parameters are the same for each quadrant, defined as follows: $h = 0.2, ext = 0.1, ei = 0.4, ii = 0.8, ir = 0.2, rs = 0.1, m_N = m_S = m_E = m_W = 0.05$.

Fig. 6 shows the fast-simulation results[10] for the model for each of the four quadrants. This functionality is built-in in the FlyFast tool. In the figure, the fractions of numbers of the components in each of the four states at each of the four locations are shown. Note that these fractions correspond to appropriate predicates on standard atomic propositions; for instance the fraction of components in state S at quandrant A is captured by $\mathtt{s} \wedge \mathtt{a}$, assuming the following declarations: $\mathtt{s}\,\mathsf{at}\,\mathtt{S}$ and $\mathtt{a}\,\mathsf{def}\,(\mathbf{my}.\mathsf{loc} = A)$. The simulation of single elements, taken as the average over 10 runs shows a very good correspondence with the fast-simulation results. The results also show good correspondence to the original SEIR model [5] when the probability to move between quadrants is set to zero and in the initial state the total population is in state S and in one specific quadrant in the former model. Besides fast simulation, that gives an idea of the

[9] We assume: `N(A) = N(B) = S(C) = S(D) = E(A) = E(D) = W(B) = W(C) = undefined`, with `[loc = undefined] =` $\perp$ for all loc .

[10] Experiments have been performed using the FlyFast on-the-fly mean field model checker on a PC with an Intel Core i7 1.7GHz, RAM 8Gb.

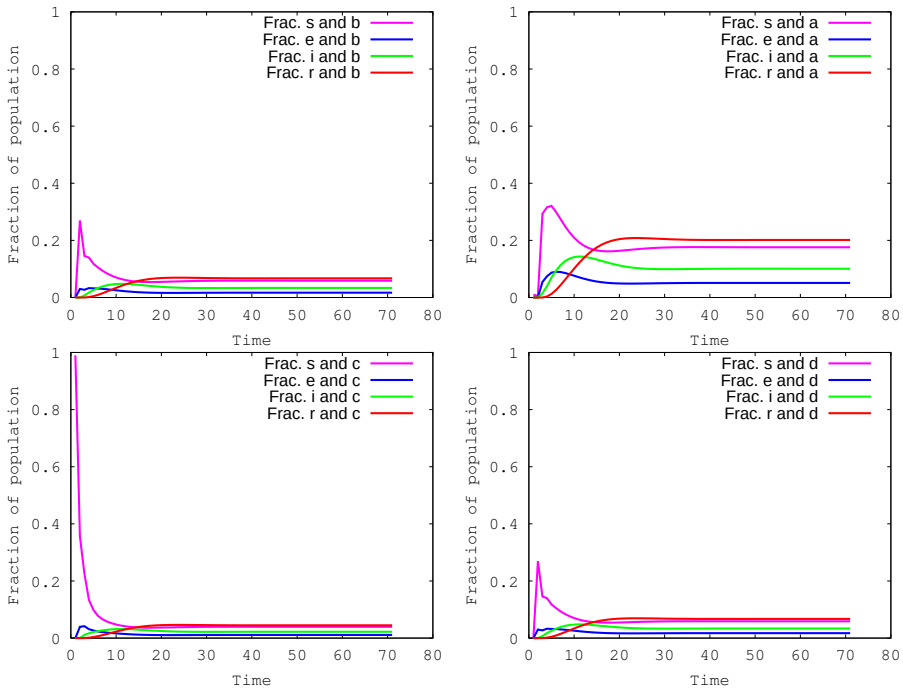

Fig. 6. Fast simulation for each of the four quadrants.

average global behaviour of the system, we can also analyse the behaviour of a single component in the context of the overall behaviour. We consider two example properties as illustration. Let us first consider a component initially in state `S` and located in A and let atomic propositions `i` and `c` be declared as follows: `i` at `I` and `c` def ($\mathbf{my}.\mathsf{loc} = C$). The following formula (P1) states that the probability is greater than p that that the component ends up infected in quadrant C by time t: $\mathcal{P}_{>p}(\mathrm{tt}\,\mathcal{U}^{\leq t}\,(i \wedge c))$. FlyFast allows one to study the dynamics of the actual probability as a function of t, by means of the notation $\mathcal{P}_{=?}(\mathrm{tt}\,\mathcal{U}^{\leq t}\,(i \wedge c))$ and the resulting graph, for the above initial conditions and for the first 70 time units is shown in Fig. 7 (left). For comparison, the formula for an agent starting in C and ending up in A and being infected is shown as well in the same figure. The results for a more complicated, nested, formula (P2) are shown in Fig. 7 (right). P2 expresses the probability, over time, of a component reaching a situation in which it is neither infected nor exposed, and from which it can reach a state in which it has a probability higher than 0.15 to be infected and located in `C` within the next 10 time units; formula P2 is given below, where `i` is assumed defined by `i` at `I`: $\mathcal{P}_{=?}(\mathrm{tt}\;\mathcal{U}^{\leq t}\,(\neg(i \vee e) \wedge \mathcal{P}_{>0.15}(\mathrm{tt}\;\mathcal{U}^{\leq 10}\,(i \wedge c))))$. The formula has been considered for a component which is initially in A and in state `S`; the figure shows also a similar formula, where the role of A and C is exchanged and a probability higher than 0.45 instead of 0.15 is considered.

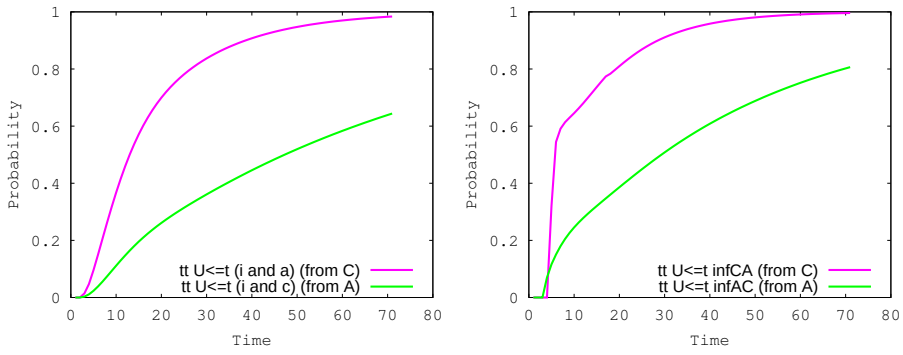

Fig. 7. Model checking results for properties P1 (left) and P2 (right).

For both types of properties a considerable difference in the probabilities can be observed for an agent that is initially located in A or in C, due to the flow of movement that has been introduced. This illustrates a clear dependence of the results on the dynamically changing spatial distribution of components. The total number of states, actions and transitions for the resulting FlyFast *object specification* is 52, 114 and 468 respectively, while the number of states of the global *approximated* model which have been generated for the analysis of formula P2 is 2.323 (2.185 when A and C are swapped). The model checking time for the more complicated nested formula P2 and for all values of t (70) is 10.343 (9.921 when A and C are swapped) ms, $\approx 148\,(141)\,ms$ per checking session, for a model with a total population of 10.100 objects. A well-known feature of mean-field model checking is that the model checking time is independent of the size of the population, however, further experimentation with more extended spatial models and more attributes, that do effect this time, is planned as future work.

5 Conclusions

The attribute-based interaction paradigm is deemed fundamental for agent interaction in the context of Autonomic or Collective Adaptive Systems [9, 20, 3, 1, 12]. In this paper we have presented a attribute-based coordination modelling language as a front-end for FlyFast, an on-the-fly mean-field model-checker for bounded PCTL. The language extends the original FlyFast modelling language by replacing its actions with input (output, respectively) actions where senders (receivers, respectively) are specified by means of predicates on dynamic attributes on system components, where a component is a process/attribute-store pair. A translation to the standard FlyFast language has been presented, its correctness has been showed as well as an example of its application to a simple case study. It should be noted that the introduction of attributes in a process model is an *intrinsic* source of complexity in terms of component state-space size. Such

an increase, in the worst case, goes with $|\mathcal{V}_\Delta|^{|Att_\Delta|} \cdot (|\mathcal{A}^O_\Delta| + 1)$, where Att_Δ is the set of attributes of the component, $\mathcal{V}_\Delta$ is the set of values they can take, and $\mathcal{A}^O_\Delta$ is the set of output actions occurring in the component specification (which may appear in its outbox). The obvious consequence of this is that one has to carefully ponder the importance and necessity of each and every new attribute used in a model, although, it must be kept in mind that the real source of state-space explosion is the size of the system, and this issue is addressed by Mean-Field approximation. A first optimisation consists in considering only reachable component states as well as eliminating actions with constant zero probability and simplifying boolean combinations of FlyFast atomic propositions in the translation. A possible additional line of investigation is the study of techniques for DTMC minimization to Mean-field analysis, so that the number of difference equations can decrease as a consequence, in a similar way as for CTMCs and the number of differential equations in fluid flow analysis [24].

References

1. Alrahman, Y.A., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: ACM SAC 2015 pp. 1840–1845.
2. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-Checking Algorithms for Continuous-Time Markov Chains. IEEE TSE 29(6), 524–541 (2003).
3. Bortolussi, L., De Nicola, R., Galpin, V., Gilmore, S., Hillston, J., Latella, D., Loreti, M., Massink, M.: CARMA: collective adaptive resource-sharing markovian agents. In: QAPL 2015, EPTCS, vol. 194, pp. 16–31 (2015).
4. Bortolussi, L., Hillston, J.: Fluid model checking. In: CONCUR. LNCS, vol. 7454, pp. 333–347. Springer-Verlag (2012).
5. Bortolussi, L., Hillston, J., Latella, D., Massink, M: Continuous approximation of collective systems behaviour: A Tutorial. Performance Evaluation. Elsevier, 70:317–349, 2013.
6. Bortolussi, L., Latella, D., Massink, M: Stochastic Process Algebra and Stability Analysis of Collective Systems. In: *COORDINATION '13*, LNCS, vol. 7890, pp. 1–15. Springer (2013).
7. Ciancia, V., Latella, D., Massink, M.: On Space in CARMA. Technical Report TR-QC-01-2015, QUANTICOL (2015).
8. Ciancia, V., Latella, D., Massink, M.: An Attribute-based Front-end for FlyFast. Technical Report CNR-ISTI 2015-TR-041. 2015.
9. 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. Software Engineering for Collective Autonomic Systems - The ASCENS Approach, LNCS, vol. 8998, pp. 3–71. Springer (2015).
10. Feng, C., Hillston, J.: PALOMA: A process algebra for located markovian agents. In: QEST 2014. LNCS, vol. 8657, pp. 265–280. Springer (2014).
11. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing. Springer-Verlag 6(5), 512–535 (1994).
12. Hillston, J., Loreti, M.: Specification and analysis of open-ended systems with Carma. In: Agent Environments for Multi-Agent Systems IV, LNCS, vol. 9068, pp. 95–116. Springer (2015).

13. John, M., Lhoussaine, C., Niehren, J.: Dynamic compartments in the imperative *pi*-calculus. In: CMSB 2009. LNCS, vol. 5688, pp. 235–250. Springer (2009).
14. John, M., Lhoussaine, C., Niehren, J., Uhrmacher, A.M.: The attributed pi calculus. In: CMSB 2008. LNCS, vol. 5307, pp. 83–102. Springer (2008).
15. Latella, D., Loreti, M., Massink, M.: On-the-fly fast mean-field model-checking. In: TGC 2013, Revised Selected Papers. LNCS, vol. 8358, pp. 297–314. Springer (2013).
16. Latella, D., Loreti, M., Massink, M.: On-the-fly Probabilistic Model Checking. In: ICE 2014. EPTCS, ISSN: 2075-2180, vol. 166, pp. 45–59 (2014).
17. Latella, D., Loreti, M., Massink, M.: On-the-fly PCTL fast mean-field approximated model-checking for self-organising coordination. Science of Computer Programming 110, 23–50 (2015).
18. Latella, D., Loreti, M., Massink, M.: Investigating Fluid-Flow Semantics of Asynchronous Tuple-Based Process Languages for Collective Adaptive Systems. In: *COORDINATION '15*, LNCS, vol. 9037, pp. 19–34. Springer (2015).
19. Latella, D.: Comunicazione basata su proprietà nei sistemi decentralizzati [Property-based inter-process communication in decentralized systems] (dec. 1983), graduation Thesis. Istituto di Scienze dell'Informazione. Univ. of Pisa (in italian).
20. Latella, D., Loreti, M., Massink, M., Senni, V.: On stocs: A stochastic extension of SCEL. Software, Services, and Systems - Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering. LNCS, vol. 8950, pp. 619–640. Springer (2015),
21. Le Boudec, J.Y., McDonald, D., Mundinger, J.: A generic mean field convergence result for systems of interacting objects. In: QEST 07. pp. 3–18. IEEE (2007).
22. Lesser, V.R., Corkill, D.D.: Functionally accurate, cooperative distributed systems. IEEE Transactions on Systems, Man, and Cybernetics 11(1), 81–96 (1981).
23. Massink, M., Latella, D.: Fluid Analysis of Foraging Ants. In: *COORDINATION '12*, LNCS, vol. 7274, pp. 152–165. Springer (2012).
24. Tschaikowski, M., Tribastone, M.: A unified framework for differential aggregations in markovian process algebra. J. Log. Algebr. Meth. Program., 84(2):238–258, 2015.