Tuple-based Coordination of Stochastic Systems with Uniform Primitives

Stefano Mariani
DISI, ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, Italy
Email: s.mariani@unibo.it

Andrea Omicini
DISI, ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, Italy
Email: andrea.omicini@unibo.it

***Abstract*—Complex computational systems – such as pervasive, adaptive, and self-organising ones – typically rely on simple yet expressive coordination mechanisms: this is why coordination models and languages can be exploited as the sources of the essential abstractions and mechanisms to build such systems. While the features of tuple-based models make them well suited for complex system coordination, they lack the probabilistic mechanisms for modelling the stochastic behaviours typically required by adaptivity and self-organisation.**

To this end, in this paper we explicitly introduce *uniform primitives* as a probabilistic specialisation of standard tuple-based coordination primitives, replacing don't know non-determinism with uniform distribution. We define their semantics and discuss their expressiveness and their impact on system predictability.

I. INTRODUCTION

While computational systems grow in complexity, coordination models and technologies are more and more essential to harness the intricacies of intra- and inter-system interaction [1], [2]. In particular, tuple-based coordination models – derived from the original LINDA [3] – have shown their power in the coordination of pervasive, adaptive, and self-organising systems [4], such as SAPERE [5] and MoK [6].

A foremost feature of computational models for open, adaptive and self-* systems is *non-determinism*. LINDA features *don't know* non-determinism in the access to tuples in tuple spaces, handled with a *don't care* approach: *(i)* a tuple space is a multiset of tuples where multiple tuples possibly match a given template; *(ii)* which tuple among the matching ones is actually retrieved by a *getter* operation (`in`, `rd`) can be neither specified nor predicted (*don't know*); *(iii)* nonetheless, the coordinated system is designed so as to keep on working whichever is the matching tuple returned (*don't care*).

The latter assumption requires that when a process uses a template matching multiple tuples, which specific tuple is actually retrieved is not relevant for that process. This is not the case, however, in many of today adaptive and self-organising systems, where processes may need to implement *stochastic behaviours* like "most of the time do this" or "not always do that"—which obviously do not cope well with don't know non-determinism. For instance, all the nature-inspired models and systems emerged in the last decade – such as chemical, biochemical, stigmergic, and field-based – are examples of the broad class of *self-organising* systems that precisely require such a sort of behaviour [7]—which by no means can be enabled by the canonical LINDA model and its direct derivatives.

To this end, in this paper we define *uniform coordination primitives* (`uin`, `urd`) – first mentioned in [8] – as the specialisation of LINDA *getter* primitives featuring *probabilistic non-determinism* instead of don't know non-determinism. Roughly speaking, uniform primitives allow programmers to both specify and (*statistically*) *predict* the probability to retrieve one specific tuple among a bag of matching tuples, thus making it possible to statistically control non-deterministic systems.

Accordingly, in this paper we first define uniform primitives based on the probabilistic framework from [9] (Section II), then demonstrate their expressive power both formally – by exploiting *probabilistic modular embedding* [10] – and by discussing some examples (Section III). Finally, we compare uniform primitives with other approaches in probabilistic and stochastic coordination (Section IV).

II. UNIFORM PRIMITIVES

LINDA *getter* primitives – that is, data-retrieval primitives `in` and `rd` – are shared by all tuple-based coordination models, and provide them with *don't know non-determinism*: when one or more tuples in a tuple space match a given template, *any* of the matching tuples can be non-deterministically returned.

In a single getter operation, only a *point-wise* property affects tuple retrieval: that is, the conformance of a tuple to the template, independently of the *spatial* context—namely, the other tuples in the same space. Furthermore, in a sequence of getter operations, don't know non-determinism makes any prediction of the overall behaviour impossible: e.g., reading one thousand times the same template in a tuple space with ten matching tuples could possibly lead to retrieving the same tuple all times, or one hundred times each, or whatever admissible combination one could think of—no prediction possible, according to the model. Again, then, only a point-wise property can be ensured even in *time*: that is, only the mere compliance to the model of each individual operation in the sequence.

Instead, uniform primitives enrich tuple-based coordination models with the ability of performing operations that ensure *global* system properties instead of point-wise ones, both in space and in time. More precisely, uniform primitives replace don't know non-determinism with *probabilistic non-determinism* to *situate* a primitive invocation in space – the tuple actually retrieved depends on the other tuples in the space – and to *predict* its behaviour in time — statistically, the distribution of the tuples retrieved will tend to be uniform, over time.

Whereas exploiting probabilistic non-determinism to its full extent would lead to the definition of the complete set of uniform coordination primitives – including, e.g., `uinp` and `urdp` primitives –, here we aim at understanding the fundamental mechanisms making tuple-based models well suited for complex system coordination, by enhancing them with the probabilistic mechanisms for modelling the stochastic behaviours typically required by adaptivity and self-organisation. Accordingly, in this paper we focus only on the two uniform primitives (`uin`, `urd`) that specialise the basic LINDA getter primitives. In the remainder of this section, first (Subsection II-A) we define them informally, then (Subsection II-B) we provide them with a formal semantic specification according to the probabilistic framework defined in [9].

A. *Informal semantics*

The main motivation behind uniform primitives is to introduce a *simple* yet *expressive* probabilistic mechanism in tuple-based coordination: simple enough to work as a specialisation of standard LINDA operations, expressive enough to model the most relevant stochastic behaviours of complex computational systems such as adaptive and self-organising ones.

Whereas expressiveness is discussed in Section III, simplicity is achieved by defining uniform primitives as specialised versions of standard LINDA primitives: so, first of all, `uin` and `urd` are compliant with the standard semantics of `in` and `rd`. In the same way as `in` and `rd`, `uin` and `urd` ask tuple spaces for one tuple matching a given template, suspend when no matching tuple is available, return a matching tuple chosen non-deterministically when one or more matching tuples are available in the tuple space. As a straightforward consequence, any tuple-based coordination system working with `in` and `rd` would also work by using instead `uin` and `urd`, respectively—and any process using `in` and `rd` could adopt `uin` and `urd` instead without any further change.

On the other hand, the nature of the specialisation lays precisely in the way in which a tuple is non-deterministically chosen among the (possibly) many tuples matching the template. While in standard LINDA the choice is performed based on don't know non-determinism, uniform primitives exploit instead *probabilistic non-determinism* with *uniform distribution*. So, if a standard getter primitive requires a tuple with template T, and m tuples $t_1,..,t_m$ matching T are in the tuple space when the request is executed, any tuple $t_{i\in\{1..m\}}$ could be retrieved, but nothing more could be said—no other assertion is possible about the result of the getter operation. Instead, when a uniform getter primitive requires a tuple with template T, and m tuples $t_1,..,t_m$ matching T are available in the tuple space when the request is served, one assertion is possible about the result of the getter operation: that is, each of the m matching tuples $t_{i\in\{1..m\}}$ has exactly the same probability $1/m$ to be returned. So, for instance, if 2 `colour(blue)` and 3 `colour(red)` tuples occur in the tuple space when a `urd(colour(X))` is executed, the probability of the tuple retrieved to be `colour(blue)` or `colour(red)`) is exactly 40% or 60%, respectively.

Operationally, uniform primitives behave as follows. When executed, a uniform primitive takes a *snapshot* of the tuple space, "freezing" its state at a certain point in time—and space, being a single tuple space the target of basic LINDA primitives. The snapshot is then exploited to assign a probabilistic value $p_i \in [0,1]$ to any tuple $t_{i\in\{1..n\}}$ in the space—where n is the total number of tuples in the space. There, non-matching tuples have value $p = 0$, matching tuples have value $p = 1/m$ (where $m \leq n$ is the number of matching tuples), and the overall sum of probability values is $\sum_{i=1..n} p_i = 1$. The choice of the matching tuple to be returned is then statistically based on the computed probabilistic values.

As a consequence, while standard getter primitives exhibit point-wise properties only, uniform primitives feature *global* properties, both in space and time. In terms of spatial context, in fact, standard getter primitives can return a matching tuple independently of the other tuples currently in the same space—so, they are "context unaware". Instead, uniform getter primitives return matching tuples based on the overall state of the tuple space—so, their behaviour is *context aware*. In terms of time, too, sequences of standard getter operations present no meaningful properties. Instead, by definition, sequences of uniform getter operations tend to globally exhibit a uniform distribution over time. So, for instance, performing N `urd(colour(X))` operations over a tuple space containing 10 `colour(white)` and 100 `colour(black)` tuples, leads to a sequence of returned tuples which, while N grows, would tend to contain ten times more `colour(black)` tuples than `colour(white)` ones.

B. *Formal semantics*

In order to define the semantics of (getter) uniform primitives, we rely upon a simplified version of the process-algebraic framework in [9], dropping multi-level priority probabilities. In detail, we exploit closure operator $\uparrow$, handles h, and closure term G as follows:

(i) handles coupled to actions (open transitions) represent tuple templates associated with primitives;

(ii) handles listed in closure term G represent tuples offered (as synchronisation items) by the tuple space (modelled as a process);

(iii) closure term G associates handles (tuples) with their cardinality in the tuple space;

(iv) closure operator $\uparrow$ *(a)* matches admissible synchronisations between processes and the tuple space, and *(b)* computes their associated probability distribution based upon handle-associated values.

It is worth to note that closure operator $\uparrow$ could be seen as following our statistical interpretation of a uniform primitive: it takes a snapshot of the tuple space state – *matching*, step *(a)* – then samples it probabilistically — *sampling*, step *(b)*.

1) Semantics of `uin` (uniform consumption): Three transition rules define the operational semantics of the `uin` primitive for *uniform consumption*:

[SYNCH-C] Open transition representing the request for process-space synchronisation upon template T, which leads to the snapshot:

$$\texttt{uin}_T.P \mid \langle t_1,..,t_n\rangle \xrightarrow{T} \texttt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{(t_1,v_1),..,(t_n,v_n)\}$$

where $v_{i=1..n} = \mu(T,t_i)$, and $\mu(\cdot,\cdot)$ is the standard matching function of LINDA, hence $\forall i, v_i ::= 1 \mid 0$.

[CLOSE-C] Closed unlabelled transition (reduction) representing the internal computation assigning probabilities to synchronisation items (uniform distribution computation):

$$\texttt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{(t_1,v_1),..,(t_n,v_n)\} \hookrightarrow \texttt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{(t_1,p_1),..,(t_n,p_n)\}$$

where $p_j = \frac{v_j}{\sum_{i=1}^{n} v_i}$ is the absolute probability of retrieving tuple t_j, with $j = 1..n$.

[EXEC-C] Open transition representing the probabilistic response to the requested synchronisation (the sampling):

$$\texttt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{..,(t_j,p_j),..\} \xrightarrow{t_j}_{p_j} P[t_j/T] \mid \langle t_1,..,t_n\rangle \backslash t_j$$

where $[\cdot/\cdot]$ represents term substitution in process P continuation, and $\backslash$ is multiset difference, expressing removal of tuple t_j from the tuple space.

2) Semantics of `urd` *(uniform reading):* As for standard LINDA getter primitives, the only difference between *uniform reading* (`urd`) and uniform consumption (`uin`) is the non-destructive semantics of the reading primitive `urd`. This is reflected by EXEC-R open transition:

[EXEC-R] The same as EXEC-C, except for the fact that it does not remove matching tuple

$$\texttt{urd}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{..,(t_j,p_j),..\} \xrightarrow{t_j}_{p_j} P[t_j/T] \mid \langle t_1,..,t_n\rangle$$

whereas other transitions are left unchanged.

3) Example: As an example, in the following system state

$$\texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle$$

where $\mu(T,tx)$ holds for $x = a,b,c$, the following synchronisation transitions are enabled:

(a) $\texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle \xrightarrow{ta}_{0.5} P[ta/T] \mid \langle ta,tb,tc\rangle$

(b) $\texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle \xrightarrow{tb}_{0.25} P[tb/T] \mid \langle ta,ta,tc\rangle$

(c) $\texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle \xrightarrow{tc}_{0.25} P[tc/T] \mid \langle ta,ta,tb\rangle$

For instance, if transition *(a)* wins the probabilistic selection, then the system evolves according to the following trace—simplified by summing up cardinalities and probabilities in order to enhance readability:

$$\texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle \xrightarrow{T} \texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle \uparrow \{(ta,2),(tb,1),(tc,1)\}$$
$$\hookrightarrow \texttt{uin}_T.P \mid \langle ta,ta,tb,tc\rangle \uparrow \{(ta,\tfrac{1}{2}),(tb,\tfrac{1}{4}),(tc,\tfrac{1}{4})\} \xrightarrow{ta}_{\frac{1}{2}} P[ta/T] \mid \langle ta,tb,tc\rangle$$

III. EXPRESSIVENESS

In [11], authors demonstrate that LINDA-based languages cannot implement probabilistic models: a LINDA process calculus, although Turing-complete, is not expressive enough to express probabilistic choice [11]. In our specific case, the gain of expressiveness is formally proven in [12], where uniform primitives are formally proven to be strictly more expressive than standard LINDA coordination primitives by exploiting *probabilistic modular embedding* (PME) [10], an extension to *modular embedding* [13] explicitly meant to capture the expressiveness of stochastic systems.

In particular, if we denote with uLINDA the LINDA coordination model where standard getter primitives `rd` and `in` are replaced with uniform getter primitives `urd` and `uin`, then uLINDA is proven to be strictly *more expressive* than LINDA according to PME, since uLINDA *probabilistically embeds* ($\succeq_p$) LINDA, but not the other way around—so that formally, according to PME, LINDA and uLINDA are not observationally equivalent ($\not\equiv_o$):

$$\text{uLINDA} \succeq_p \text{LINDA},\ \text{LINDA} \not\succeq_p \text{uLINDA} \implies \text{uLINDA} \not\equiv_o \text{LINDA}$$

Since formally asserting a gap in expressiveness does not necessarily make it easy for the reader to fully appreciate how much this can make the difference for adaptive and self-organising systems, in the remainder of this section we discuss two examples showing how uniform primitives make it possible to *(i)* have some self-organising property appear by emergence (Subsection III-A), and *(ii)* straightforwardly design stochastic systems reproducing some simple yet meaningful nature-inspired behavioural pattern, such as pheromone-based coordination (Subsection III-B).

```java
LogicTuple templ;
while(!die){
  templ = LogicTuple.parse("ad(S)");
  // Pick a server probabilistically
  op = acc.urd(tid, templ, null);
  // Plain Linda version
  // op = acc.rd(tid, templ, null);
  if (op.isResultSuccess()) {
    service = op.getLogicTupleResult();
    // Submit request
    req = LogicTuple.parse(
      "req("+service.getArg(0)+","+reqID+")"
    );
    acc.out(tid, req, null);
  }
}
```

Fig. 1. Java code for clients looking for services.

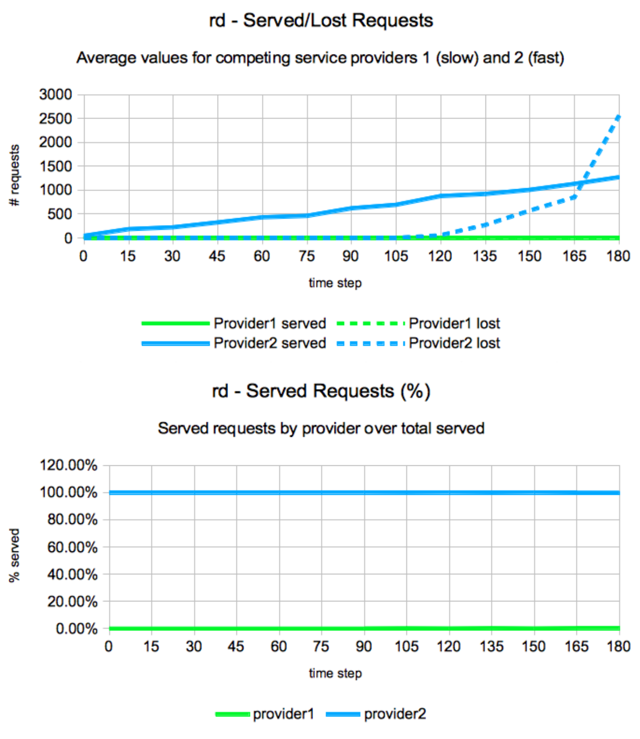

Fig. 2. Clients using `rd` primitive: service provider 1 is under-exploited.

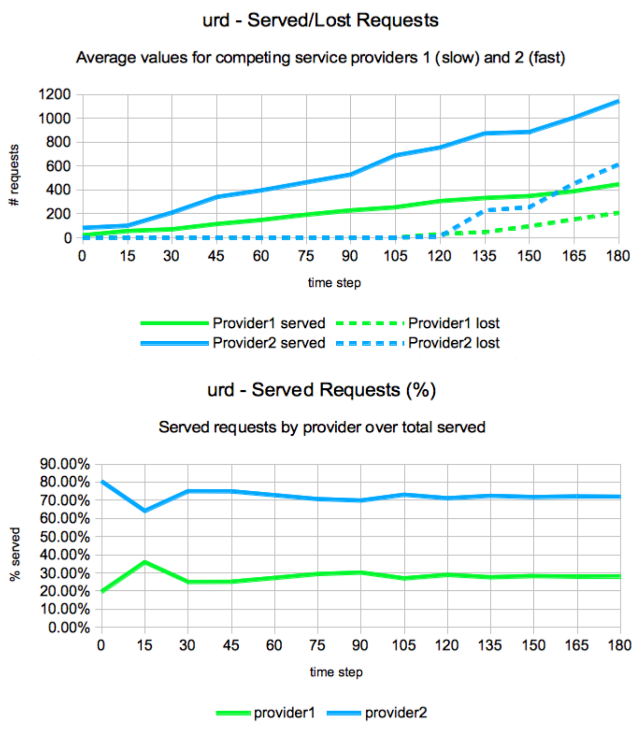

Fig. 3. Clients using `urd` primitive: a certain degree of *fairness* is guaranteed, based on self-organisation.

A. Load Balancing

In order to better explain what the "basic mechanisms enabling self-organising coordination" actually are – that is, a minimal construct able (alone) to impact the observable properties of a coordinated system – we discuss the following scenario: two service providers are both offering the same service to clients – through proper "advertising tuples" –; the first is slower than the second, that is, it needs more time to process a request—thus modelling differences in computational power.

Their working cycle is quite simple: a worker thread gets requests from a shared tuple space, then puts them in the master thread (the actual service provider) bounded queue. The master thread continuously polls the queue looking for requests to serve: when one is found, it is served, then the master emits another advertising tuple; if none is found, the master does something else, then re-polls the queue—no advertising is done. The decoupling enforced by the queue is useful to model the fact that service providers should not block on the space waiting for incoming requests, so as to be free of performing other jobs meanwhile—e.g. reporting, resource clean-up, etc. The queue is bounded to model memory constraints.

In this setting, clients (whose Java code is listed in Fig. 1) search for available services first via `rd` primitive (Fig. 2), then via `urd` (Fig. 3). All charts' values are not single runs, but average values resulting from different runs—e.g., value plotted at time step 60 is not that of a single run, but the average of the number of requests observable at time step 60 of a number of runs (actually, 30).

By using the `rd` primitive we *blindly commit* to the actual implementation of the LINDA model currently at hand. For instance, Fig. 2 gives some hints about the implementation used for our simulation – the TuCSoN coordination middleware [14], [15] –: since provider 1 is almost unused, we understand that `rd` is implemented as a FIFO queue, always matching the first tuple among many ones—provider 2 advertising tuple, in this case. The point here is that such a prediction was not possible prior to the simulation, and with no information about the actual LINDA implementation used.

By using primitive `urd` instead (Fig. 3), we know – and can predict – how much each service provider will be exploited by clients: since we know by design that after successfully serving a request a provider emits an advertising tuple, and that such tuples are those looked for by clients, we know that the faster provider will produce more tuples, hence it will be more frequently found than the slower one. From Fig. 3 charts, in fact, we can see how the system of competing service providers self-organises by splitting incoming requests. Furthermore, such split is not statically designed or superimposed, but results by *emergence* from a number of run-time factors, such as clients interactions, service providers computational load, computational power, and memory. It should also be noted that such form of *load balancing* is not the only benefit gained when using `urd` over `rd`: actually, the `urd` simulation successfully serves $\simeq$ 1600 requests – distributed among providers 1 and 2 according to uniform primitive semantics – losing $\simeq$ 600, whereas the `rd` simulation serves successfully $\simeq$ 1250 – leaving provider 1 unused – losing over 2500.

B. Pheromone-based coordination

In *pheromone-based coordination* used by ants to find optimal paths – as well as by many ant-inspired computational systems, such as [16], [17] – each agent basically wanders

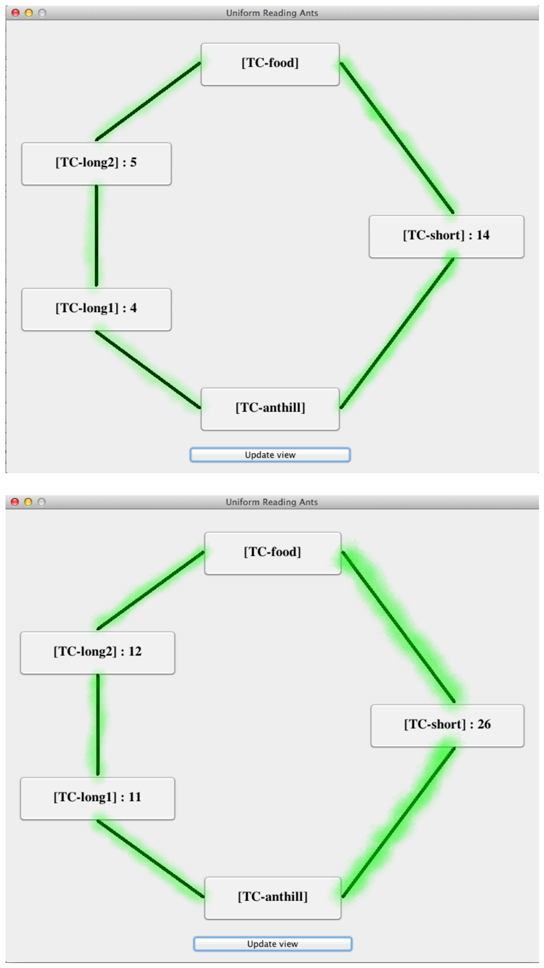

Fig. 4. Digital ants search food *(top box)* wandering randomly from their anthill *(bottom box)*.

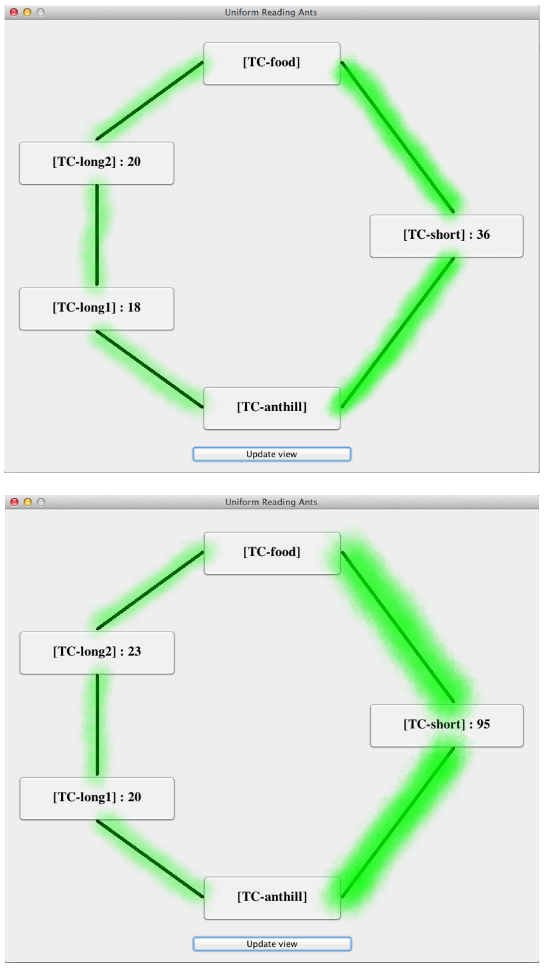

Fig. 5. By `urd`-ing digital pheromones left while carrying food, digital ants stochastically find the optimal path toward the food source.

randomly through the network until it finds a pheromone trail, which the agent is *likely* to follow based on the trail "strength".

Here, aspects such as pheromone release, scent, and evaporation [16] are not relevant: instead, the above-mentioned notions of "randomness" and "likelihood" are on the one hand *essential* for pheromone-based coordination, on the other hand *require* uniform primitives to be designed using a tuple-based coordination model. In particular, we consider a network of n nodes representing places p_i, with $i = 1..n$, through which ant agents walk. The default tuple space in node p_i contains at least one *neighbour* tuple $\texttt{n(}p_j\texttt{)}$ for each neighbour node p_j and the neighbourhood relation is reflexive—so, if node p_i and p_j are neighbours, p_i tuple space contains tuple $\texttt{n(}p_j\texttt{)}$ and p_j tuple space contains tuple $\texttt{n(}p_i\texttt{)}$. Pheromone deposit in node p_i is modelled by the insertion of a new tuple $\texttt{n(}p_i\texttt{)}$ in every neighbour node p_i.

Thus, ants wandering through places and ants following trails can both be easily modelled using uniform primitives: ant agents just need to look locally for neighbour tuples through a `urd(n(P))`. If no pheromone trail is to be detected nearby, every neighbour place is represented by a single tuple, so all neighbour places have the same probability to be chosen—thus leading to random wandering of ants. In case some of the neighbours contains a detectable trail, the corresponding neighbour tuple occurs more than once in the local tuple space: so, by using uniform primitives, the tuple corresponding to a neighbour place with a pheromone trail has a greater probability to be chosen than the others.

For instance, say `p1`, `p2`, `p3` are neighbour places. Without a pheromone trail, an ant in `p1` moves to either `p2` or `p3` with the same probability, starting from the following system state:

$$\texttt{urd(n(X))}.P \mid \langle \texttt{n(p2),n(p3)} \rangle$$

There, the enabled synchronisation transitions are

(a) $\texttt{urd(n(X))}.P \mid \langle \texttt{n(p2),n(p3)} \rangle$
$\xrightarrow{\texttt{n(p2)}}_{0.5}$
$P[\texttt{p2/X}] \mid \langle \texttt{n(p2),n(p3)} \rangle$

(b) $\texttt{urd(n(X))}.P \mid \langle \texttt{n(p2),n(p3)} \rangle$
$\xrightarrow{\texttt{n(p3)}}_{0.5}$
$P[\texttt{p3/X}] \mid \langle \texttt{n(p2),n(p3)} \rangle$

that is, an ant agent in `p1` has the same probability (50%) to move to either `p2` or `p3`—which exactly models random ant wandering.

Instead, if a pheromone trail involves p3 – so that for instance p1 contains 2 tuples n(p3) – the initial system state would be

$$\texttt{urd(n(X))}.P \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$$

There, the enabled synchronisation transitions are

(c) $\texttt{urd(n(X))}.P \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$
$\xrightarrow{\texttt{n(p2)}}_{0.\overline{3}}$
$P[\texttt{p2/X}] \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$

(d) $\texttt{urd(n(X))}.P \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$
$\xrightarrow{\texttt{n(p3)}}_{0.\overline{6}}$
$P[\texttt{p3/X}] \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$

which exactly models the fact that the ant agent in p1 is more likely to move to p3 than to p2, thus (probabilistically) following the pheromone trail.

A crucial point, here, is to understand the issue of system *predictability* with / without uniform primitives. Reachable states for the system above would not change by replacing urd with rd: the transitions above would work in the same way *apart from* probabilistic labelling. This essentially means that a standard LINDA coordinated system would potentially reach the same states as the one with uniform primitives: the point is, nevertheless, that quantitative information would be available for the latter system, not for the former.

In particular, in the second example above, the reachable states are *(c)* $P[\texttt{p2/X}] \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$ and *(d)* $P[\texttt{p3/X}] \mid \langle \texttt{n(p2),n(p3),n(p3)} \rangle$. Using urd, we know that states *(c)* and *(d)* would be reached with probability $.\overline{3}$ and $.\overline{6}$, respectively: so, both a probabilistic prediction on the single system run, and a statistic prediction over multiple system runs are made possible by the use of uniform primitives. The usage of rd, instead, allows for nothing similar: we just know that both states *(c)* and *(d)* could be reached, but no quantitative predictions of any sort are possible.

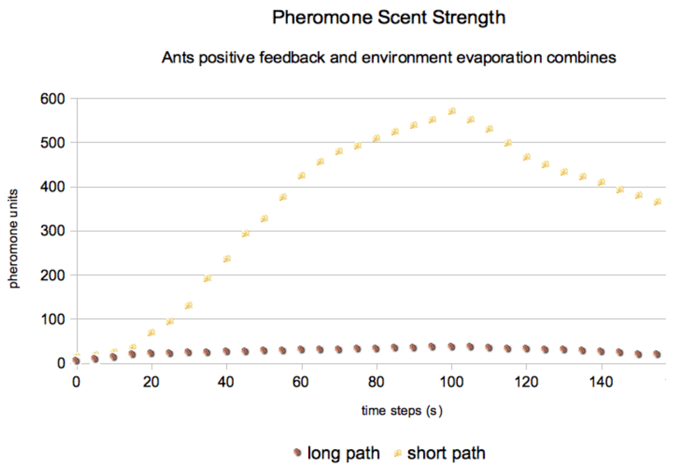

Fig. 6. Pheromones strength across time. Descending phase corresponds to food depletion in food tuple centre: no new pheromones added, evaporation makes strength decline.

Our experiments are conducted in a toy scenario involving digital ants and pheromones programmed in ReSpecT [18] upon the TuCSoN coordination middleware [14]. The experiment involves ten digital ants starting from the anthill with the goal of finding food, and follows the "canonical" assumptions of ant systems. So, at the beginning, any path has equal probability of being chosen, thus modelling random walking of ants in absence of pheromone. As ants begin to wander around, eventually they find food, and release pheromone on their path while coming back home. As a consequence, the shortest path eventually gets more pheromone since it takes less time to travel on it rather than on the longest path. Pheromones as well as connections between tuple centres are modelled as described above, with "neighbour" tuples: the more neighbour tuples of a certain type, the more likely ants will move to that neighbour tuple centre with their next step.

Fig. 4 and Fig. 5 depict a few screenshots of our toy scenario: there, five distributed tuple centres (the larger boxes) model a topology connecting the anthill *(bottom box)* to a food source *(top box)*: the leftmost path is longer, whereas the rightmost is shorter. The green "spray-like" effect on paths (black lines) models the strength of the pheromone scent: the greater and greener the path, the more pheromones lay on it.

By plotting pheromones strength evolution over time, Fig. 6 simply shows how our expectations about digital ants behaviour are met: in fact, despite starting from the situation in which any path is equi-probable (the amount of pheromones on the shortest path is the same as on the longest path), eventually the system detects the shortest path, which becomes the most exploited—and contains in fact more pheromone units.

In the Java code describing the behaviour of ants (Fig. 7), in particular in method smellPheromone() (line 10), usage of the uniform primitive urd is visible on line 27, whereas line 29 shows the tuple template given as its argument, that is, n(NBR): at runtime, NBR unifies with a TuCSoN tuple centre identifier, making it possible for the ant to move there. Quite obviously, the idea here is not just showing a new way to model ant-like systems. Instead, the example above is meant to point out how a non-trivial behaviour – that is, dynamically solving a shortest path problem – can be achieved by simply substituting uniform primitives to traditional LINDA getter primitives—which instead would not allow the system to work as required. Furthermore, the solution is adaptive, fully distributed, and based upon local information solely – thus, it appears by *emergence* –, and robust against topology changes—a ReSpecT specification implementing evaporation was used, although not shown for the lack of space.

IV. RELATED WORKS

Uniform primitives were first used in [19] as a tool for solving a specific coordination problem, called *collective sort*: however, neither there, nor in subsequent papers [20], [8], they were given but a few lines of informal definition, and their general role in the coordination of complex computational systems was not yet clarified.

In [21], similar primitives are presented and formally defined to forge the *biochemical tuple space* notion, leading a tuple space to act as a chemical simulator. There, tuples are enriched with an *activity/pertinency value* – similarly to the quantitative information defined in [11] – to resemble chemical concentrations, therefore LINDA primitives are necessarily refined with the ability to consider such numerical label. So, the main point of difference w.r.t. therein defined primitives is that *(i)* here we rely on tuples multiplicity to model probability, leaving the LINDA tuples structure untouched, *(ii)* uniform primitives are scheduled and executed as LINDA classical getter primitives, while in [21] their primitives have a *stochastic rate of execution* equipped.

To the best of our knowledge, proposals presented to extend LINDA with probabilities follow two main approaches [22]:

- *data-driven* models, where the quantitative information required to model probability is associated with the data items – the tuples – in the form of *weights*. This approach is adopted in `ProbLinCa` [11], the probabilistic version of a LINDA-based process calculus.
- *schedule-driven* models, where the quantitative information is added to the processes using special "probabilistic schedulers". This is the approach taken by [22] to define a probabilistic extension of the KLAIM model named PKLAIM.

Instead, our approach belongs to a third, novel category – which we call *interaction-driven* – where probabilistic behaviour is *(i)* associated to communication primitives – thus, neither to processes (or schedulers), nor to tuples – and *(ii)* enacted during the interaction between a process and the coordination medium—that is, solely through such primitives.

Also, uniform primitives can be seen as complementary to both the approaches taken in `ProbLinCa` and PKLAIM, where the basic LINDA model is changed quite deeply. Uniform primitives, instead, extend LINDA by *specialising* standard LINDA primitives, without changing neither tuple structure nor scheduling policy. Furthermore, uniform primitives could be used to emulate both approaches: tuple weights could be reified by their multiplicity in the space, whereas probabilistic scheduling could be obtained by properly synchronising processes upon probabilistic consumption of shared tuples. Moreover, uniform coordination primitives could be used in place of LINDA standard ones without affecting the model, merely refining don't care non-determinism as probabilistic non-determinism: as a result, all the expressiveness results and all the applications based on the canonical LINDA model do still hold using `uin` and `urd` instead of `in` and `rd`.

More complex coordination models exist in literature for which uniform primitives could play a key role in providing the probabilistic mechanisms required for the engineering of stochastic systems like adaptive and self-organising ones.

STOKLAIM [23] is an extension to KLAIM in which process actions are equipped with *rates* affecting execution probability, and execution *delays* as well—that is, time needed to carry out an action. By reifying action rates as tuples in the space, with multiplicity proportional to rates, uniform-reading such tuples would allow to probabilistically schedule actions' execution *à la* STOKLAIM. Furthermore, delays could be emulated, too, by uniform-reading a set of "time tuples", where a higher value corresponds to a lower action rate.

SAPERE [5] is a biochemically-inspired model for the engineering of complex self-organising and adaptive *pervasive service ecosystems*, where agents share *LSAs* (Live Semantic Annotation), which could be thought of as a special kind of tuples, representing them in shared contexts, and allowing them to interact and pursue their own goals. LSAs are managed through *eco-laws*, which are some sort of chemical-like rules, scheduled according to their rates Hence, uniform primitives could play in SAPERE the same role as in STOKLAIM—once eco-laws are reified as tuples with a multiplicity proportional to execution rate. Furthermore, from the pool of all LSAs which can participate in a eco-law, the ones actually consumed by the law – as *chemical reactants* – are selected probabilistically. Once again, such behaviour could be enabled by uniform consumption of reactant LSAs in eco-laws.

V. CONCLUSION

In this paper we formally define *uniform primitives* as simple specialisation of standard LINDA coordination primitives, exploiting probabilistic non-determinism in place of don't know non-determinism. We argue that uniform primitives introduce a simple yet powerful mechanism enhancing tuple-based coordination with the ability to express and predict stochastic behaviours, thus to design complex coordinated systems featuring adaptiveness and self-organisation.

```
while (!stopped) {
  if (!carryingFood) {
    // If not carrying food
    isFood = smellFood();
    if (isFood) {
      // pick up food if any
      pickFood();
    } else {
      // or stochastically follow pheromone
      direction = smellPheromone();
      move(direction);
    }
  } else {
    // If carrying food
    if (isAnthill()) {
      // drop food if in anthill
      dropFood();
    } else {
      // or move toward anthill
      direction = smellAnthill();
      move(direction);
    }
  }
}

private LogicTuple smellPheromone() {
  ITucsonOperation op = acc.urd(
    tcid,
    LogicTuple.parse("n(NBR)"),
    TIMEOUT
  );
  if (op.isResultSuccess()) {
    return op.getLogicTupleResult();
  }
}
```

Fig. 7. Java code for ants.

Acknowledgments

This work has been supported by the EU-FP7-FET Proactive project SAPERE – Self-aware Pervasive Service Ecosystems, under contract no. 256874.

References

[1] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Communications of the ACM*, vol. 35, no. 2, pp. 97–107, 1992. [Online]. Available: http://dx.doi.org/10.1145/129630.129635

[2] P. Wegner, "Why interaction is more powerful than algorithms," *Communications of the ACM*, vol. 40, no. 5, pp. 80–91, May 1997. [Online]. Available: http://dx.doi.org/10.1145/253769.253801

[3] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985. [Online]. Available: http://dx.doi.org/10.1145/2363.2433

[4] A. Omicini and M. Viroli, "Coordination models and languages: From parallel computing to self-organisation," *The Knowledge Engineering Review*, vol. 26, no. 1, pp. 53–59, Mar. 2011, special Issue 01 (25th Anniversary Issue). [Online]. Available: http://dx.doi.org/10.1017/S026988891000041X

[5] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. Di Marzo, M. Risoldi, A.-E. Tchao, S. Dobson, G. Stevenson, Y. Ye, E. Nardini, A. Omicini, S. Montagna, M. Viroli, A. Ferscha, S. Maschek, and B. Wally, "Self-aware pervasive service ecosystems," *Procedia Computer Science*, vol. 7, pp. 197–199, Dec. 2011, proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11). [Online]. Available: http://dx.doi.org/10.1016/j.procs.2011.09.006

[6] S. Mariani and A. Omicini, "Molecules of Knowledge: Self-organisation in knowledge-intensive environments," in *Intelligent Distributed Computing VI*, ser. Studies in Computational Intelligence, G. Fortino, C. Bădică, M. Malgeri, and R. Unland, Eds., vol. 446. Springer, 2013, pp. 17–22, 6th International Symposium on Intelligent Distributed Computing (IDC 2012), Calabria, Italy, 24-26 Sep. 2012. Proceedings. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32524-3_4

[7] A. Omicini, "Nature-inspired coordination models: Current status, future trends," *ISRN Software Engineering*, vol. 2013, 2013, article ID 384903, Review Article. [Online]. Available: http://dx.doi.org/10.1155/2013/384903

[8] L. Gardelli, M. Viroli, M. Casadei, and A. Omicini, "Designing self-organising MAS environments: The collective sort case," in *Environments for MultiAgent Systems III*, ser. LNAI, D. Weyns, H. V. D. Parunak, and F. Michel, Eds. Springer, May 2007, vol. 4389, pp. 254–271, 3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71103-2_15

[9] M. Bravetti, "Expressing priorities and external probabilities in process algebra via mixed open/closed systems," *Electronic Notes in Theoretical Computer Science*, vol. 194, no. 2, pp. 31–57, 16 Jan. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2007.11.003

[10] S. Mariani and A. Omicini, "Probabilistic modular embedding for stochastic coordinated systems," in *Coordination Models and Languages*, ser. LNCS, C. Julien and R. De Nicola, Eds. Springer, 2013, vol. 7890, pp. 151–165, 15th International Conference (COORDINATION 2013), Florence, Italy, 3–6 Jun. 2013. Proceedings. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38493-6_11

[11] M. Bravetti, R. Gorrieri, R. Lucchi, and G. Zavattaro, "Quantitative information in the tuple space coordination model," *Theoretical Computer Science*, vol. 346, no. 1, pp. 28–57, 23 Nov. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2005.08.004

[12] S. Mariani and A. Omicini, "Probabilistic embedding: Experiments with tuple-based probabilistic languages," in *28th ACM Symposium on Applied Computing (SAC 2013)*, Coimbra, Portugal, 18–22 Mar. 2013, pp. 1380–1382, Poster Paper. [Online]. Available: http://dx.doi.org/10.1145/2480362.2480621

[13] F. S. de Boer and C. Palamidessi, "Embedding as a tool for language comparison," *Information and Computation*, vol. 108, no. 1, pp. 128–157, 1994. [Online]. Available: http://dx.doi.org/10.1006/inco.1994.1004

[14] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sep. 1999, special Issue: Coordination Mechanisms for Web Agents. [Online]. Available: http://dx.doi.org/10.1023/A:1010060322135

[15] TuCSoN, "Home page," 2013. [Online]. Available: http://tucson.unibo.it

[16] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, Jul. 2004. [Online]. Available: http://mitpress.mit.edu/books/ant-colony-optimization

[17] H. V. D. Parunak, S. Brueckner, and J. Sauter, "Digital pheromone mechanisms for coordination of unmanned vehicles," in *1st International Joint Conference on Autonomous Agents and Multiagent systems*, C. Castelfranchi and W. L. Johnson, Eds., vol. 1. New York, NY, USA: ACM, 15–19 Jul. 2002, pp. 449–450. [Online]. Available: http://dx.doi.org/10.1145/544741.544843

[18] A. Omicini and E. Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, Nov. 2001. [Online]. Available: http://dx.doi.org/10.1016/S0167-6423(01)00011-9

[19] M. Casadei, L. Gardelli, and M. Viroli, "Simulating emergent properties of coordination in Maude: the collective sort case," in *5th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2006)*, ser. Electronic Notes in Theoretical Computer Science, C. Canal and M. Viroli, Eds. CONCUR 2006, Bonn, Germany: Elsevier Science B.V., 31 Aug. 2006, pp. 59—80. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2007.05.022

[20] M. Casadei, M. Viroli, and L. Gardelli, "On the collective sort problem for distributed tuple spaces," *Science of Computer Programming*, vol. 74, no. 9, pp. 702–722, 2009, Special Issue on the 5th International Workshop on Foundations of Coordination Languages and Architectures (FOCLASA '06). [Online]. Available: http://dx.doi.org/10.1016/j.scico.2008.09.018

[21] M. Viroli and M. Casadei, "Biochemical tuple spaces for self-organising coordination," in *Coordination Languages and Models*, ser. LNCS, J. Field and V. T. Vasconcelos, Eds. Lisbon, Portugal: Springer, Jun. 2009, vol. 5521, pp. 143–162, 11th International Conference (COORDINATION 2009), Lisbon, Portugal, Jun. 2009. Proceedings. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02053-7_8

[22] A. Di Pierro, C. Hankin, and H. Wiklicky, "Probabilistic Linda-based coordination languages," in *3rd International Conference on Formal Methods for Components and Objects (FMCO'04)*, ser. LNCS, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Berlin, Heidelberg: Springer, 2005, vol. 3657, pp. 120–140. [Online]. Available: http://dx.doi.org/10.1007/11561163_6

[23] R. De Nicola, D. Latella, J.-P. Katoen, and M. Massink, "StoKlaim: A stochastic extension of Klaim," Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" (ISTI), Tech. Rep. 2006-TR-01, 2006. [Online]. Available: http://www1.isti.cnr.it/˜Latella/StoKlaim.pdf