Tuple-based Coordination of Stochastic Systems with Uniform Primitives

Stefano Mariani Andrea Omicini
{s.mariani, andrea.omicini}@unibo.it

DISI
Alma Mater Studiorum—Università di Bologna

WOA 2013
Torino, Italy
2 December 2013

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Outline

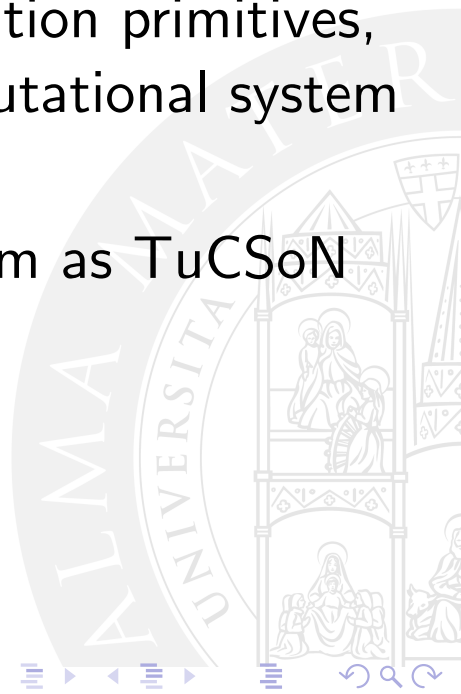
1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Non-determinism, Coordination & Stochastic Behaviour

- A foremost feature of computational models for *open*, *adaptive*, and *self-** systems is non-determinism
- Non-determinism is the basic mechanisms for stochastic behaviour
- Since most of the complexity featured by such systems depends on the interaction among components, coordination models should feature *non-deterministic mechanisms* for *stochastic behaviour*

Goals of the Research

- Devising out the *basic mechanisms* for stochastic coordination
- Finding a *minimal* set of primitives for most (all) of the most relevant stochastic systems
- Embedding such mechanisms as *tuple-based* co-ordination primitives, in order to address the general need of complex computational system engineering
- Defining their formal semantics and implementing them as TuCSoN primitives



Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Don't Care Non-determinism

- LINDA features *don't know* non-determinism handled with a *don't care* approach:

  don't know which tuple among the matching ones is retrieved by a getter operation (`in`, `rd`) can be neither specified nor predicted

  don't care nonetheless, the coordinated system is designed so as to keep on working whichever is the matching tuple returned

- Instead, adaptive and self-organising systems require stochastic behaviours like *"most of the time do this"*, *"sometimes do that"*
- Possibly with some quantitative specification of *"most of the time"* and *"sometimes"*

→ As it is, non-determinism in tuple-based models does not fit the need of stochastic behaviour specification

LINDA "Local" Nature – In Time & Space

No context — In a single getter operation, only a *local*, point-wise property affects tuple retrieval: that is, the conformance of a tuple to the template, independently of the *spatial* context

- in fact, standard getter primitives return a matching tuple independently of the other tuples currently in the same space—so, they are *"context unaware"*

No history — Furthermore, in a sequence of getter operations, don't know non-determinism makes any prediction of the overall behaviour impossible. Again, then, only a point-wise property can be ensured even in *time*

- sequences of standard getter operations present no meaningful distribution over time

Outline

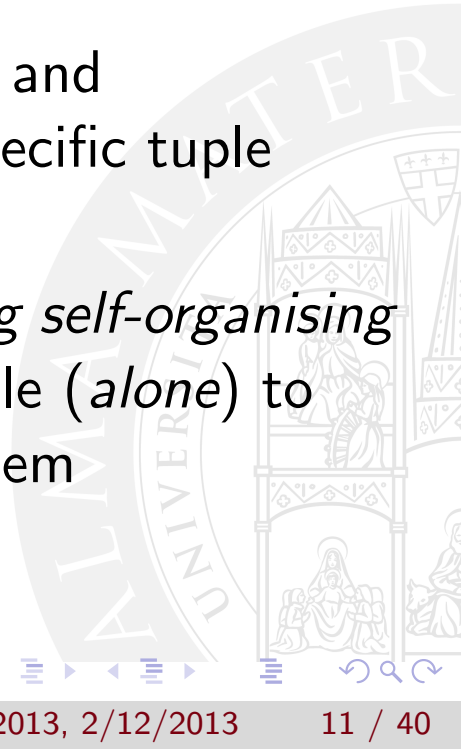
1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

uLinda: Probabilistic Non-determinism

- We define uniform coordination primitives (`uin`, `urd`) – first mentioned in [Gardelli et al., 2007] – as the *specialisation* of Linda getter primitives featuring probabilistic non-determinism instead of don't know non-determinism
- We call the new model uLinda
- Uniform primitives allow programmers to both specify and (statistically) predict the probability to retrieve one specific tuple among a bag of matching tuples
- Uniform primitives are the *"basic mechanisms enabling self-organising coordination"*—that is, a minimal set of constructs able (*alone*) to impact the observable properties of a coordinated system



uLinda: "Global" Nature

Situation & prediction

Uniform primitives replace don't know non-determinism with *probabilistic non-determinism* to

- situate a primitive invocation in space
  - uniform getter primitives return matching tuples based on the other tuples in the space—so, their behaviour is *context-aware*
- predict its behaviour in time
  - sequences of uniform getter operations tend to globally exhibit a *uniform distribution* over time

Outline

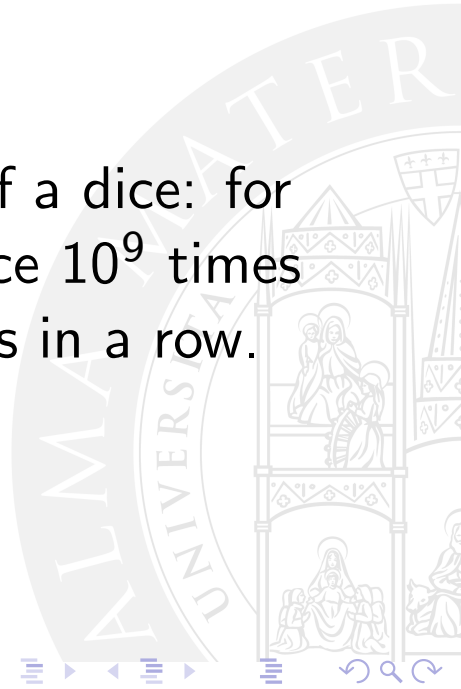
1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

LINDA: How to Roll a Dice?

- We define tuple space `dice`
- We represent a six-face dice as a collection of six tuples: `face(1)`, `...`, `face(6)`
- We roll a dice by `rd`-ing a `face/1` tuple from `dice`:

```
dice ?  rd(face(X))
```

! We do *not* obtain the overall (stochastic) behaviour of a dice: for instance, it may reasonably happen that rolling the dice 10^9 times *always* results in `X / 1`—that is, we get "1" 10^9 times in a row.



uLinda: How to Roll a Dice?

- Again, we define tuple space `dice`
- Again, we represent a six-face dice as a collection of six tuples: `face(1)`, ..., `face(6)`
- We roll a dice by `urd`-ing a `face/1` tuple from `dice`:

```
dice ? urd(face(X))
```

! Now, we *do* obtain the overall (stochastic) behaviour of a dice:

context — at every roll, the six faces of the dice `X / 1`, ..., `X / 6` have the same *probability* $P = 1/6$ to be selected

history — in the overall, repeating several times a roll, the six faces will tend to converge towards a uniform distribution

Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Informally

Operationally, uniform primitives behave as follows:

1. 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
2. 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
3. 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 $\sum_{i=1..n} p_i = 1$
4. The choice of the matching tuple to be returned is then statistically based on the computed probabilistic values

Formally I

[!] In order to define the semantics of (getter) uniform primitives, we rely upon a simplified version of the process-algebraic framework in [Bravetti, 2008], in particular the $\uparrow$ operator, dropping multi-level priority probabilities.

uin semantics

$$[\textsc{Synch-C}]\quad \mathtt{uin}_T.P \mid \langle t_1,..,t_n\rangle \xrightarrow{T} \mathtt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{(t_1,v_1),..,(t_n,v_n)\}$$

$$[\textsc{Close-C}]\quad \mathtt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{(t_1,v_1),..,(t_n,v_n)\} \hookrightarrow \mathtt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{(t_1,p_1),..,(t_n,p_n)\}$$

$$[\textsc{Exec-C}]\quad \mathtt{uin}_T.P \mid \langle t_1,..,t_n\rangle \uparrow \{..,(t_j,p_j),..\} \xrightarrow{t_j}_{p_j} P[t_j/T] \mid \langle t_1,..,t_n\rangle \backslash t_j$$

Formally II

[!] 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—transition EXEC-R.

urd semantics

[SYNCH-C] $\mathtt{uin}_T.P \mid \langle t_1, .., t_n \rangle \xrightarrow{T} \mathtt{uin}_T.P \mid \langle t_1, .., t_n \rangle \uparrow \{(t_1, v_1), .., (t_n, v_n)\}$

[CLOSE-C] $\mathtt{uin}_T.P \mid \langle t_1, .., t_n \rangle \uparrow \{(t_1, v_1), .., (t_n, v_n)\} \hookrightarrow \mathtt{uin}_T.P \mid \langle t_1, .., t_n \rangle \uparrow \{(t_1, p_1), .., (t_n, p_n)\}$

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

Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

uLinda vs Linda

In [Bravetti et al., 2005], authors demonstrate that Linda-based languages *cannot* implement probabilistic models.

PME proof

The gain in expressiveness brought by uLinda is formally proven in [Mariani and Omicini, 2013a], where uniform primitives are shown to be strictly more expressive than standard Linda primitives according to probabilistic modular embedding (PME) [Mariani and Omicini, 2013b].

In particular

$$\text{uLinda} \succeq_p \text{Linda} \quad \wedge \quad \text{Linda} \not\succeq_p \text{uLinda} \implies \text{uLinda} \not\equiv_o \text{Linda}$$

where

- $\succeq_p$ stands for "probabilistically embeds"
- $\equiv_o$ means "(PME) observational equivalence"

Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Load Balancing I

- Two service providers are offering the same service to clients through "advertising tuples"
- *Provider*1 is slower than *Provider*2 in processing requests—modeling different computational power
- Their working cycle is quite simple:
  - a worker thread gets requests from a shared tuple space putting them in the master thread bounded queue—modeling memory constraints
  - the master thread polls the queue for pending requests:
    - if 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
- Clients search for available services in two different ways:
  - for the first experiment, using `rd` primitive
  - for the second, using `urd`

Load Balancing II

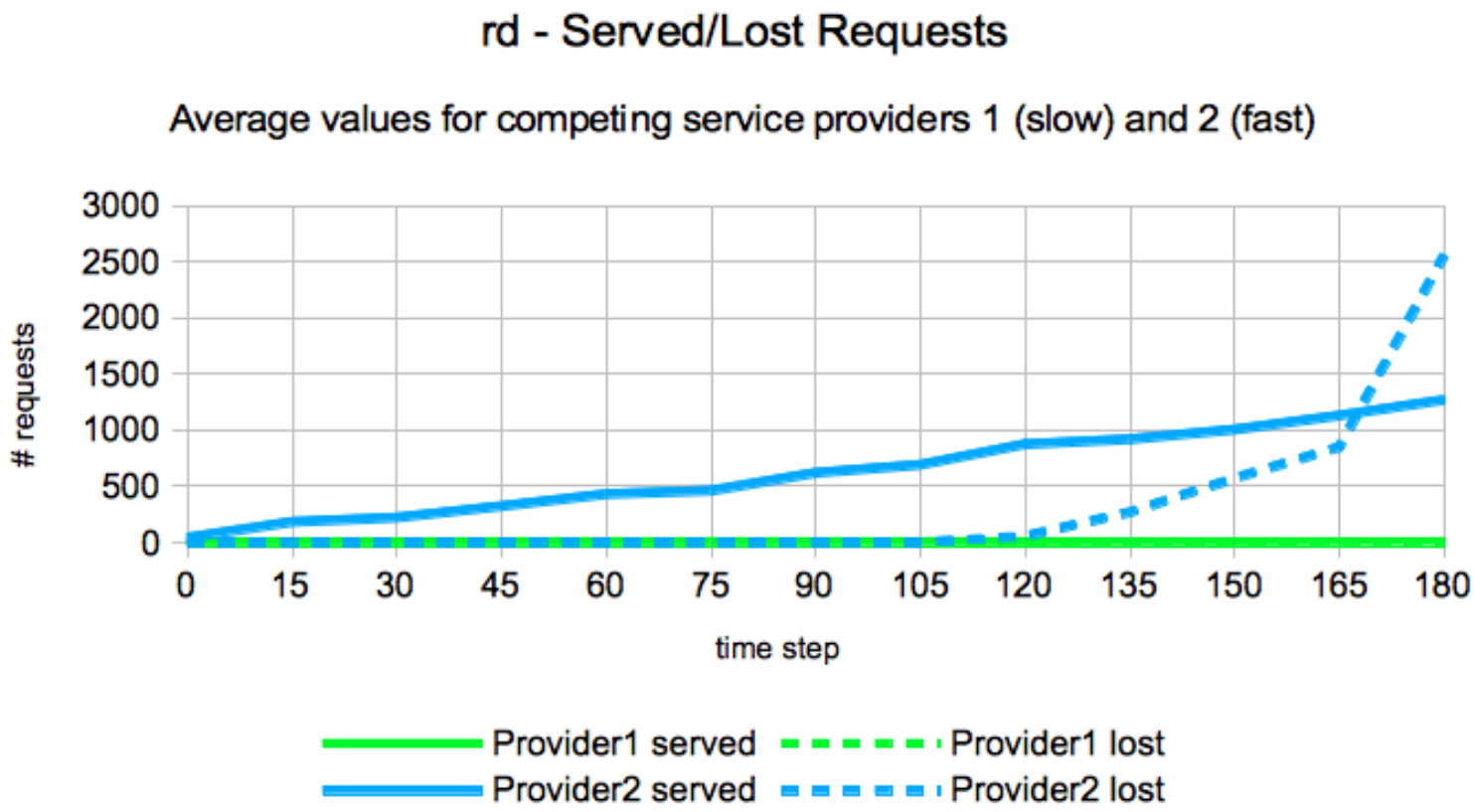


Figure : Clients using `rd`: *Provider*1 is under-exploited—actually, not at all.

Load Balancing III

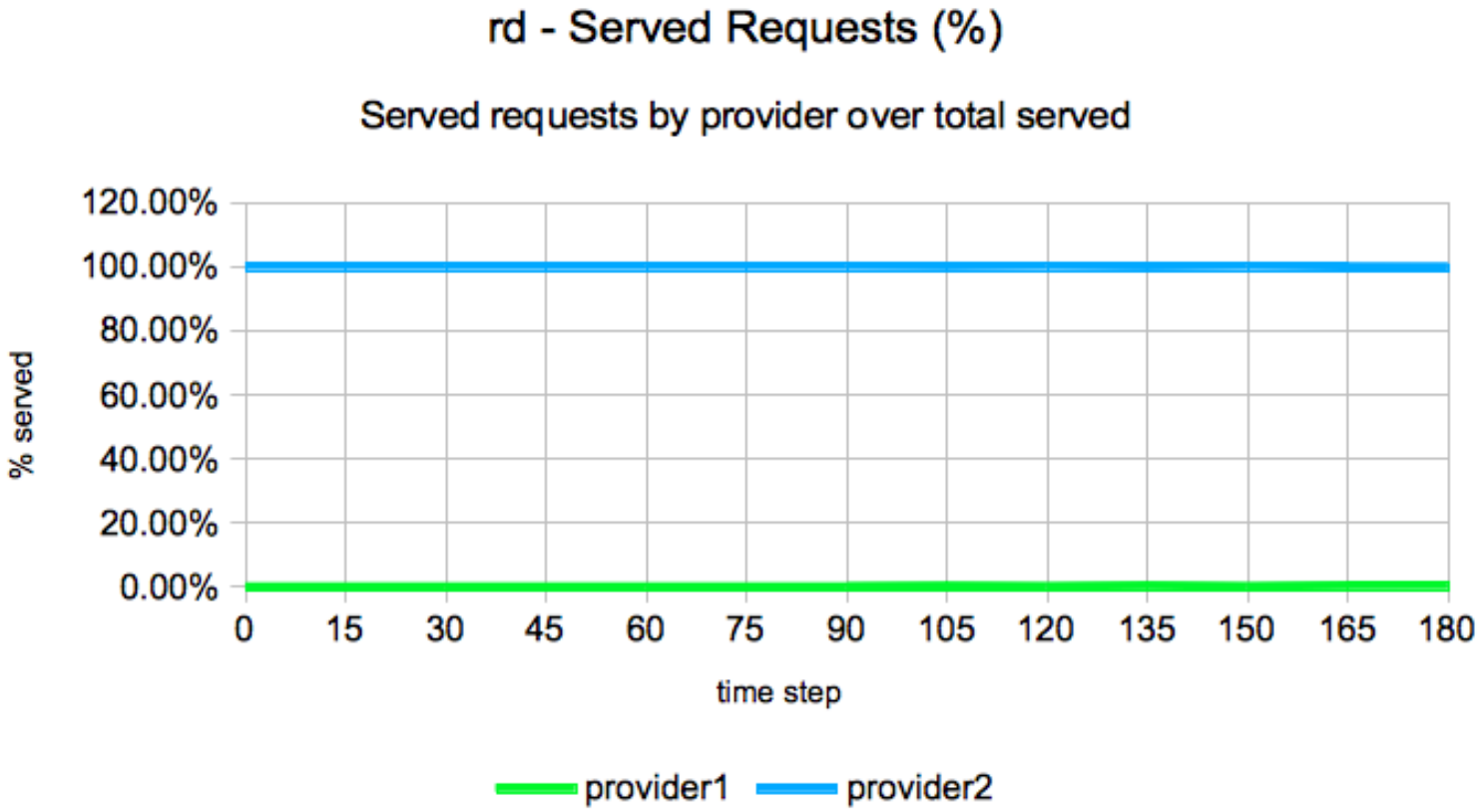


Figure : *Provider*1 and *Provider*2 exhibit no collaboration—no "load balancing".

Load Balancing IV

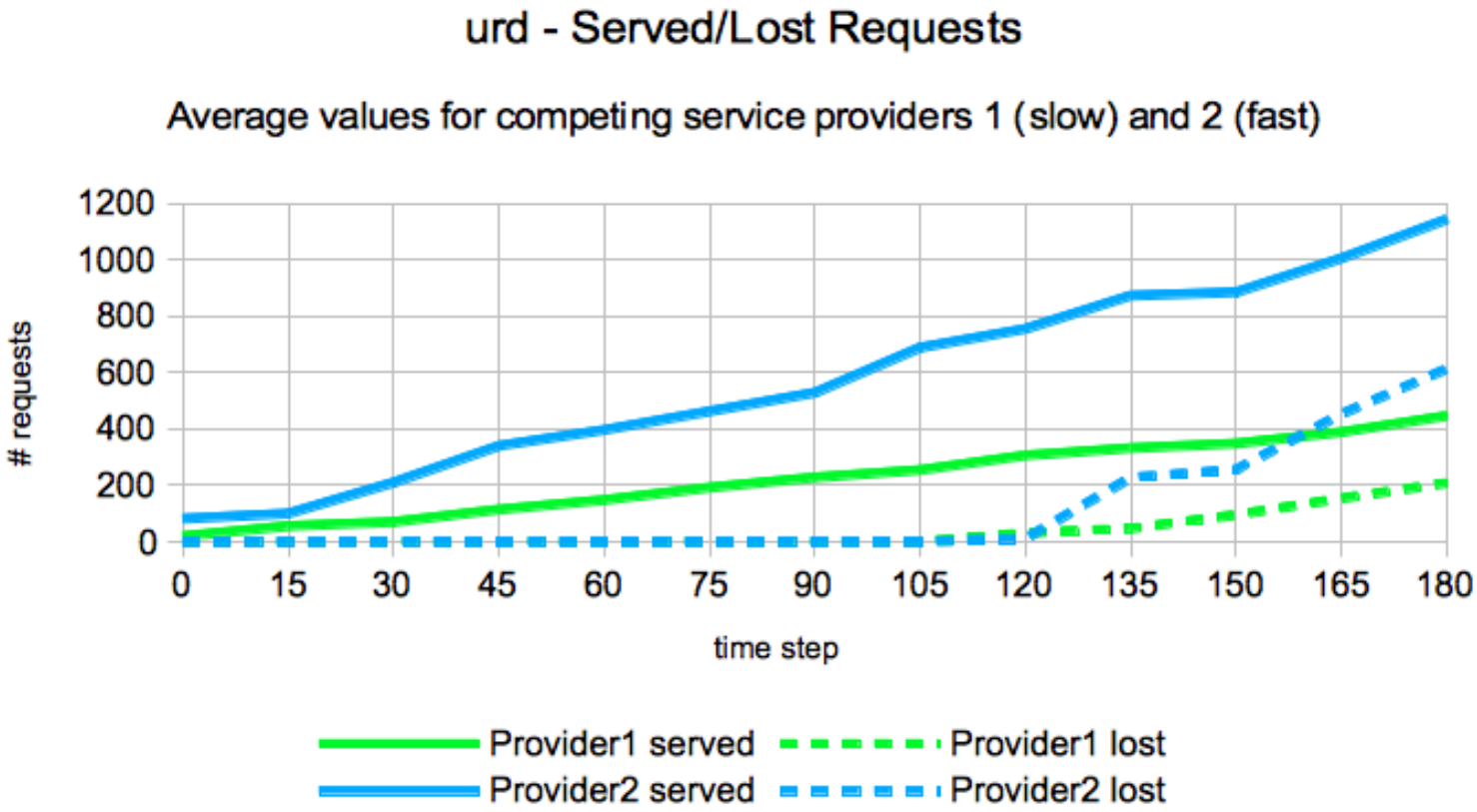


Figure : Clients using `urd`: *Provider*1 is exploited as much as it can afford.

Load Balancing V

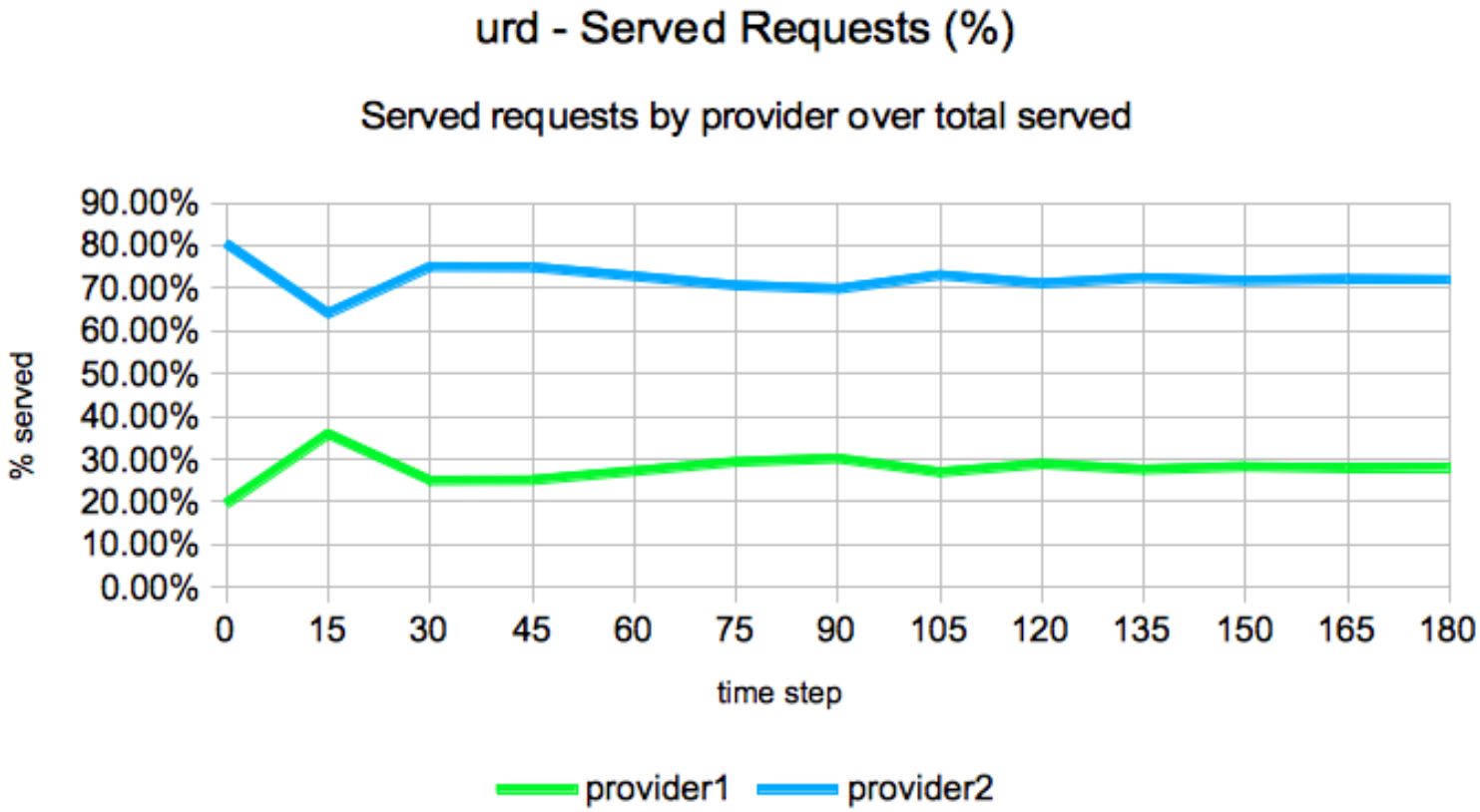


Figure : *Provider*1 and *Provider*2 exhibit some form of "load balancing", achieved by self-organisation.

Load Balancing VI

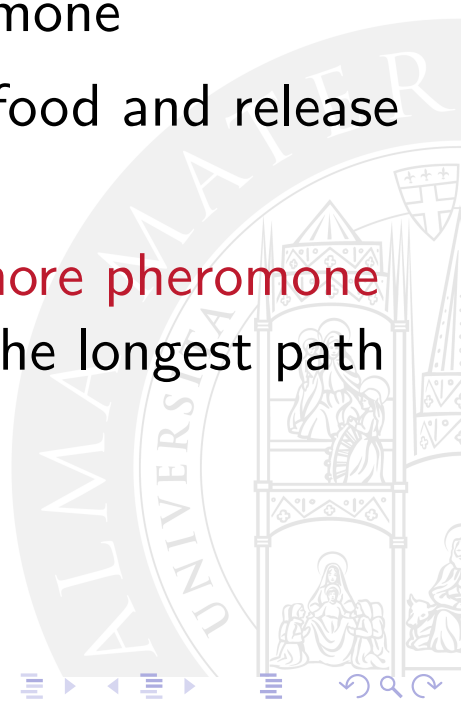
? Why such a different behaviour with *just a change of one primitive*?

- By using the `rd` primitive we *blindly commit* to the actual implementation of the LINDA model currently at hand, hence no prediction is possible prior to simulation—and with no information about implementation details
- By using primitive `urd` instead, we can predict how much each service provider will be exploited:
  - ! each provider emits an advertising tuple for each served request
  - ! such tuples are those looked for by clients
  - ! *the faster provider will produce more tuples*

  → hence it will be more frequently found than the slower one
- Furthermore, the load balancing exhibited is *not statically designed* or superimposed, but results by emergence

Pheromone-based Coordination I

- The experiment involves ten digital ants starting from the anthill with the goal of finding food
- At the beginning, any path has equal probability of being chosen, modeling random walking of ants in absence of pheromone
- As ants begin to wander around, eventually they find food and release pheromone on their way 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



Pheromone-based Coordination II

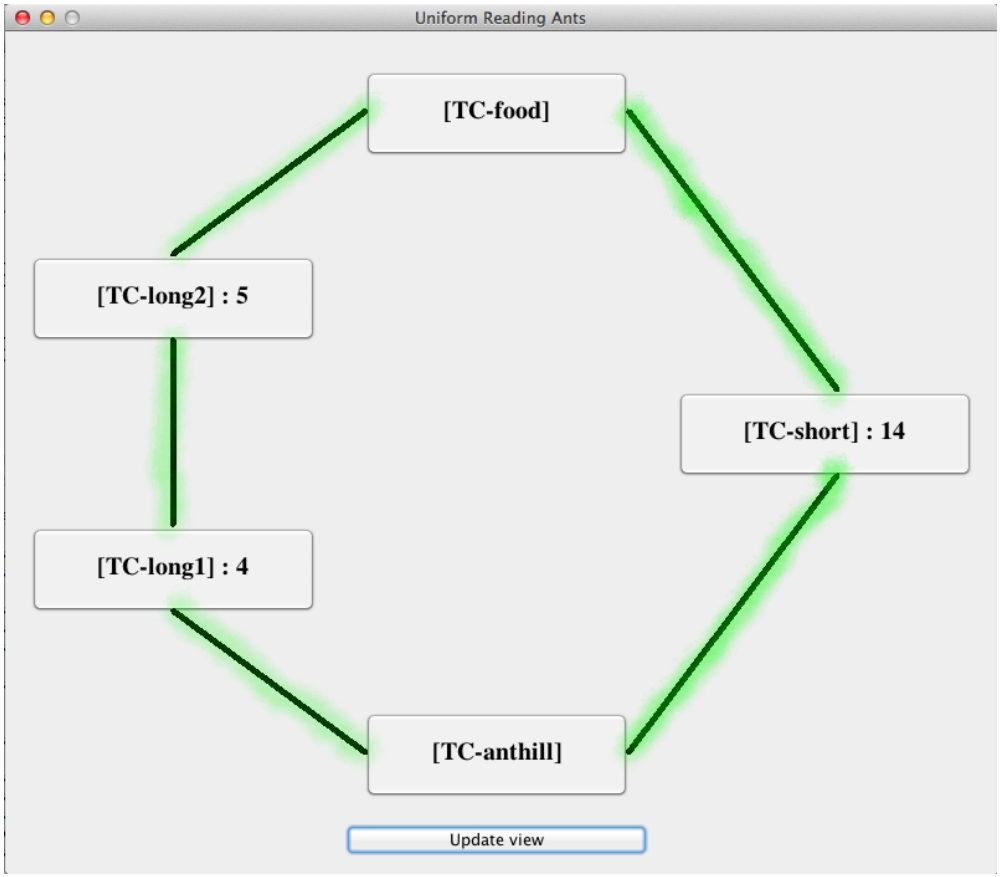


Figure : Digital ants search for food (top box) wandering randomly from their anthill (bottom box).

Pheromone-based Coordination III

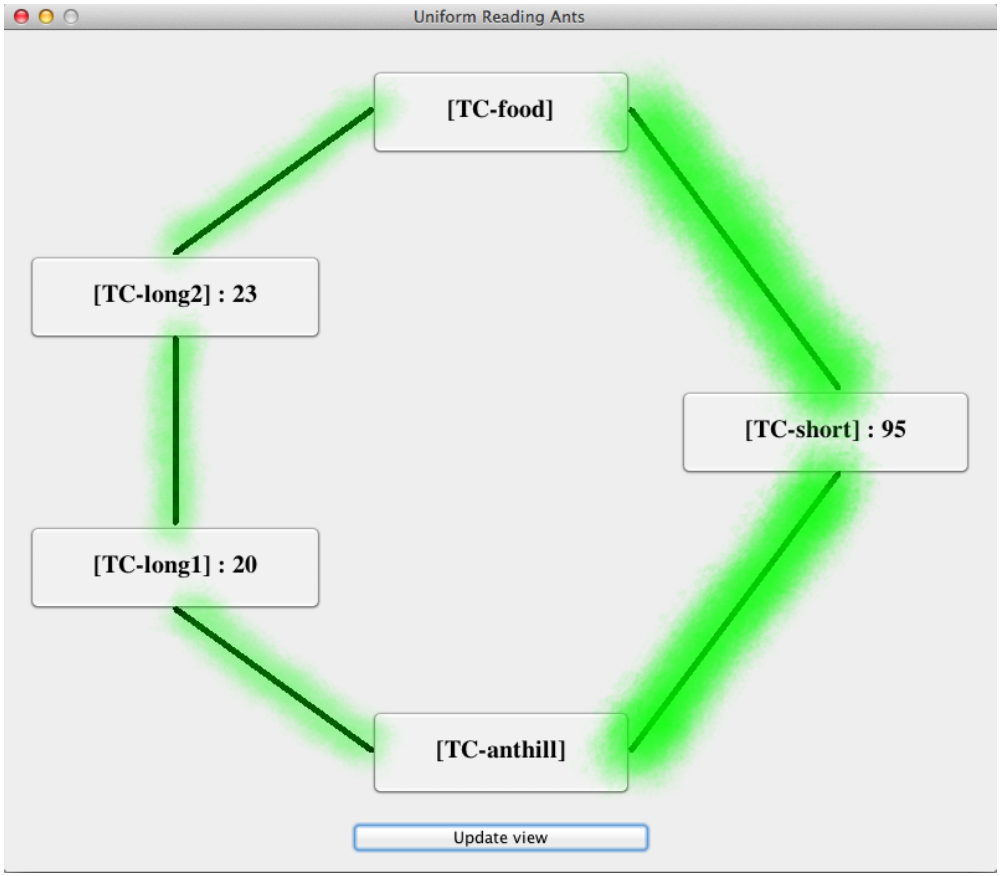


Figure : By `urd`-ing digital pheromones, ants find the optimal path toward the food source—as a self-organizing, distributed process.

Pheromone-based Coordination IV

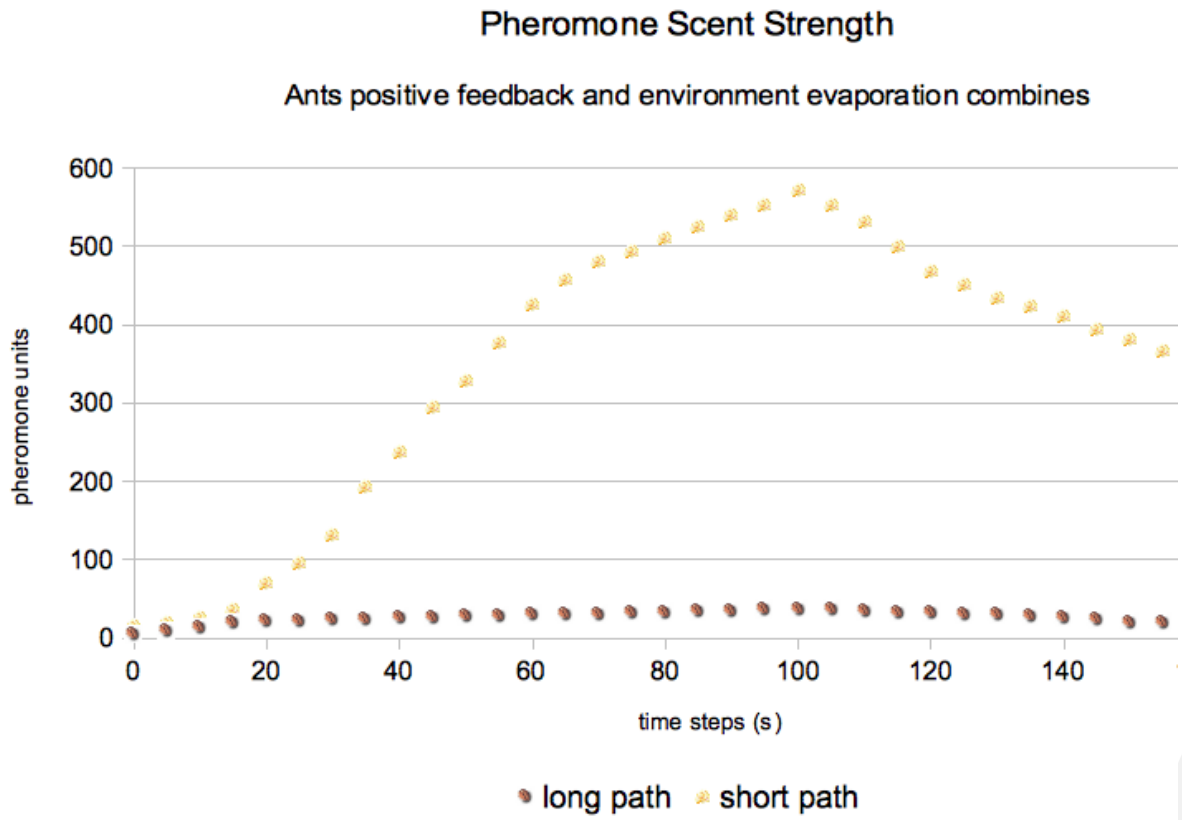


Figure : Pheromones strength across time. Descending phase corresponds to the lack of pheromone reinforcement contrasting evaporation—due to food depletion.

Pheromone-based Coordination V

- Quite obviously, the idea here is not just showing a new way to model ant-like systems
- Instead, the example is meant to highlight 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.

Outline

1. Motivations & Goals
2. Non-determinism in Tuple-based Models
3. Uniform Primitives
   - Definition & properties
   - Basic example
   - Semantics
4. Expressiveness
   - Formally
   - Informally
5. Conclusion, Ongoing & Future Work

Conclusion

- We formally define uLinda with *uniform primitives* as a specialisation of standard Linda getter primitives
- We prove the gain in expressiveness of uLinda with respect to Linda
- We claim uniform primitives can work as the basic mechanisms for *stochastic coordination*

Ongoing & Future Work

- Current TuCSoN implementation already features uniform primitives
- We will test the expressiveness of uniform primitives against all the most relevant adaptive & self-organising patterns
- We will test the expressiveness of TuCSoN uniform primitives against all the most relevant stochastic coordination models
- We will work on the efficiency of the implementation

References I

Bravetti, M. (2008).

Expressing priorities and external probabilities in process algebra via mixed open/closed systems.

Electronic Notes in Theoretical Computer Science, 194(2):31–57.

Bravetti, M., Gorrieri, R., Lucchi, R., and Zavattaro, G. (2005).

Quantitative information in the tuple space coordination model.

Theoretical Computer Science, 346(1):28–57.

Gardelli, L., Viroli, M., Casadei, M., and Omicini, A. (2007).

Designing self-organising MAS environments: The collective sort case.

In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 254–271. Springer.

3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

References II

Mariani, S. and Omicini, A. (2013a).

Probabilistic embedding: Experiments with tuple-based probabilistic languages.

In *28th ACM Symposium on Applied Computing (SAC 2013)*, pages 1380–1382, Coimbra, Portugal.

Poster Paper.

Mariani, S. and Omicini, A. (2013b).

Probabilistic modular embedding for stochastic coordinated systems.

In Julien, C. and De Nicola, R., editors, *Coordination Models and Languages*, volume 7890 of *LNCS*, pages 151–165. Springer.

15th International Conference (COORDINATION 2013), Florence, Italy, 3–6 June 2013. Proceedings.

Tuple-based Coordination of Stochastic Systems with Uniform Primitives

Stefano Mariani Andrea Omicini
{s.mariani, andrea.omicini}@unibo.it

DISI
Alma Mater Studiorum—Università di Bologna

WOA 2013
Torino, Italy
2 December 2013