Self-Organizing Approaches to System Coordination

Matteo Casadei

Alma Mater Studiorum – Università di Bologna

m.casadei@unibo.it

Progetto di Sistemi Informatici LS

A.Y. 2009/10
June 15, 2010

Outline

Motivation

New coordination models recently emerged

- Computational-fields in TOTA for pervasive computing [Mamei and Zambonelli, 2004]
- Biologically-inspired clustering of tuples in SwarmLinda [Menezes and Tolksdorf, 2004]
- Biologically-inspired pheromone infrastructures [Parunak et al., 2002]
- Infrastructures for service ecosystems [Zambonelli and Viroli, 2008]

A common view

- Addressing openness and dynamism of today and future networks
- The coordination space should not be "inert"..
- ..but rather it should self-organise to tackle adaptiveness

Self-Organising Coordination [Viroli et al., 2009]

How should coordination laws be designed?

1. They should be "local"
2. They should be continuously fired
3. They should be stochastic

What are the goals of self-organisation?

- Making some global pattern/behaviour emerge
- Leading to intrinsic adaptiveness properties

How to find good coordination laws to this end?

- Several attempts to find a methodology
- But nothing better than take inspiration from nature

Outline

Biochemical Tuple Spaces

We introduce the "biochemical tuple space" model

- a tuple space augmented with chemical reactions
- population of tuples evolves exactly as would happen in chemistry
- relying upon ideas of Computational Systems Biology

Motivations, applications

- emerging networks call for self-organising coordination
- we show an application scenario of service ecosystems

Biochemical Tuple Spaces

Main idea

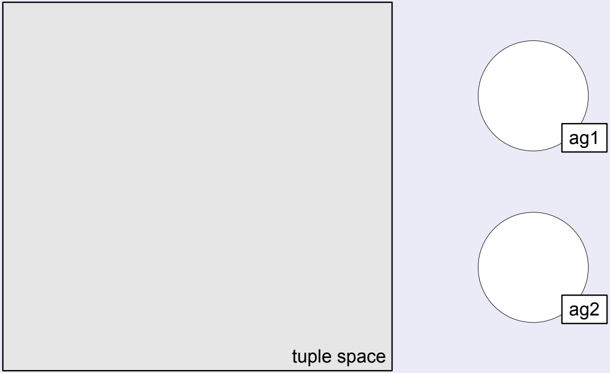
- Tuple spaces + (bio)chemical reactions as coordination laws
- Tuples have a concentration (a.k.a. weight, or activity value) as in ProbLinCa [Bravetti et al., 2004]
- Concentration is evolved “exactly” as in chemistry [Gillespie, 1977]
- Some reactions can even fire a tuple from one space to another

Why design coordination with biochemical metaphor?

- Chemistry fits coordination (Gamma) [Bonâtre and Le Métayer, 1996]
- Can get inspiration from natural/artificial biochemistry
- Can model population evolution (prey-predator, [Berryman, 1992])

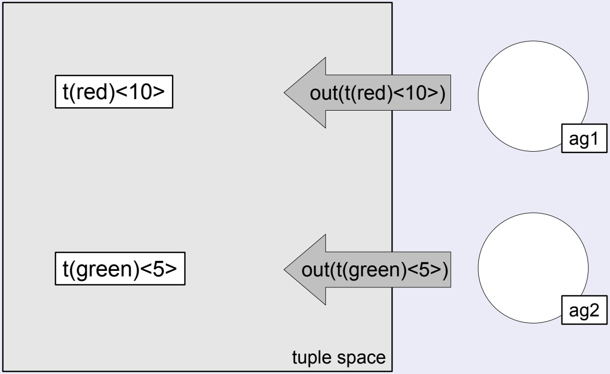
First settings

One tuple space, two agents



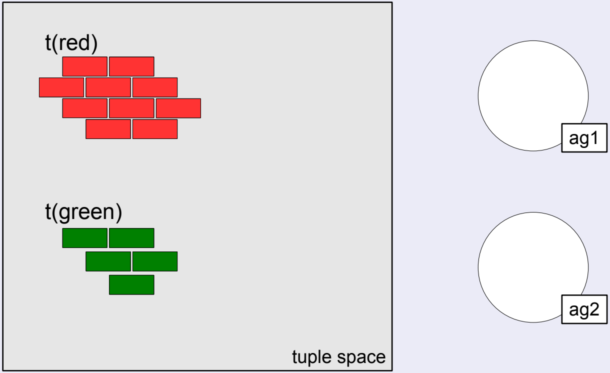
Inserting tuples

Primitive `out`: default concentration is 1



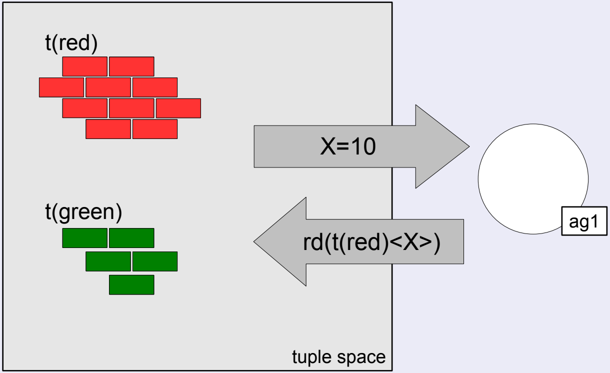
A pictorial representation

A tuple as substance of uniform molecules – but still a single tuple



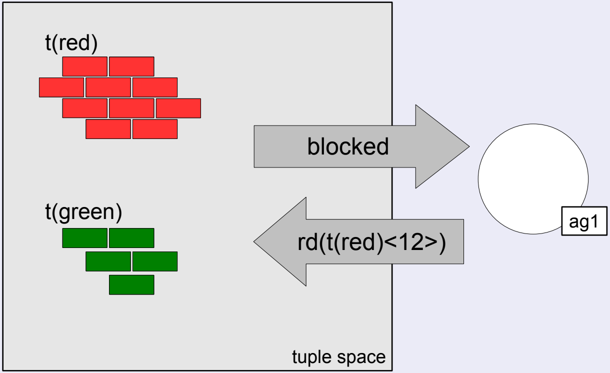
Reading Tuples

Primitive `rd`: reading current concentration



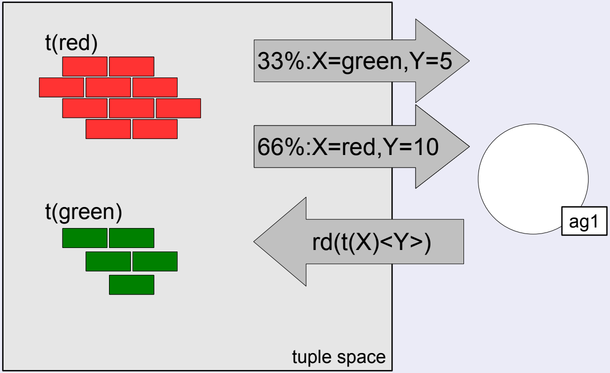
Reading Tuples

Primitive `rd`: reading a given amount – possibly blocking



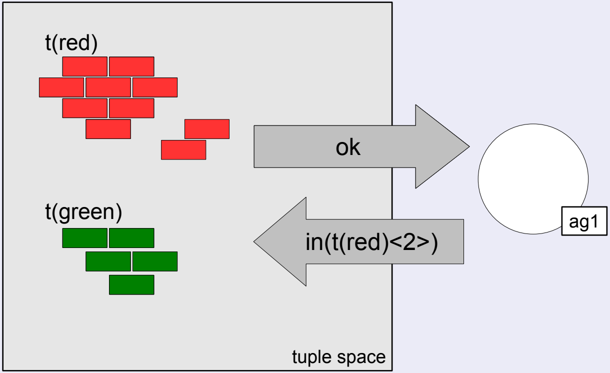
Reading Tuples

Primitive `rd`: concentration as probability, i.e., relevance



Removing Tuples

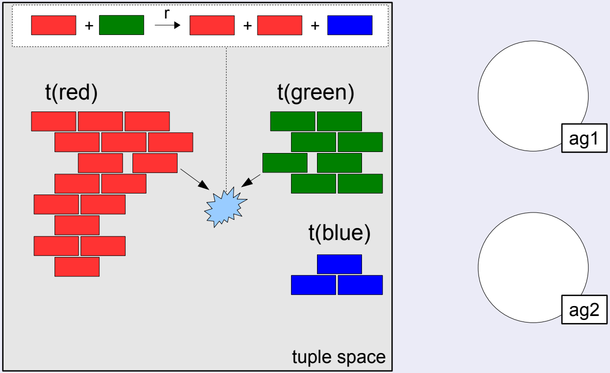
Primitive `in`: removing entirely or partially a tuple



Installing Chemical Reactions

A chemical reaction, with tuples in place of molecules

$$\mathtt{t(red)} + \mathtt{t(green)} \xrightarrow{r} \mathtt{t(red)} + \mathtt{t(red)} + \mathtt{t(blue)}$$

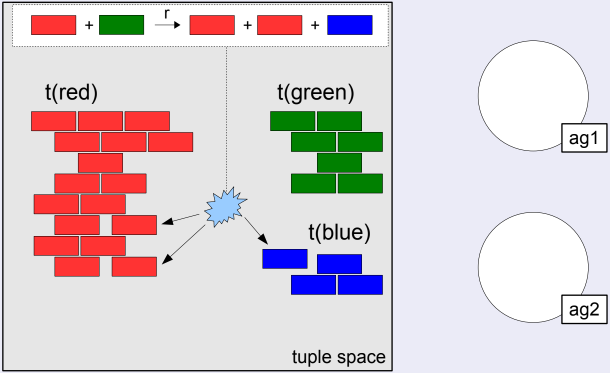


Firing Chemical Reactions

Reactions are executed over time according to [Gillespie, 1977]

$$\mathtt{t(red)} + \mathtt{t(green)} \xrightarrow{r} \mathtt{t(red)} + \mathtt{t(red)} + \mathtt{t(blue)}$$

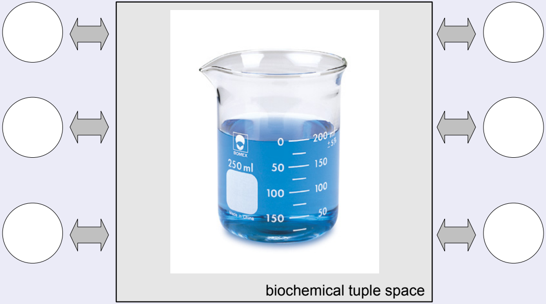
Transition (Markovian) rate: $r * \#\mathtt{t(red)} * \#\mathtt{t(green)}$



A tuple space as a chemical solution

Coordination through an exact chemical solution of tuples

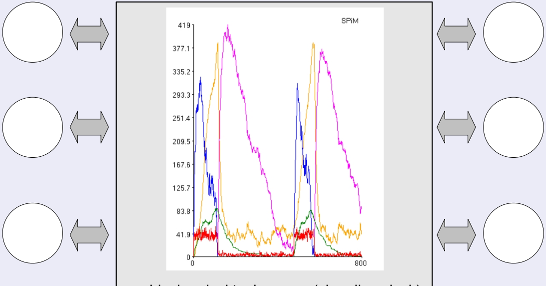
- The tuple space resembles a chemical solution in a glass
- Each tuple resembles a chemical substance
- Agents observe, insert and remove substances
- Tuple concentration drives the selection of chemical reactions



A tuple space as a chemical solution

Coordination through an exact chemical solution of tuples

- The tuple space resembles a chemical solution in a glass
- Each tuple resembles a chemical substance
- Agents observe, insert and remove substances
- Tuple concentration drives the selection of chemical reactions

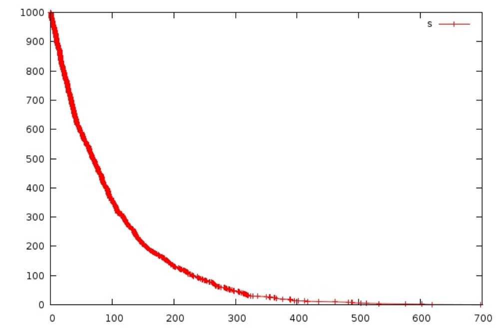


biochemical tuple space (circadian clock)

Decay example

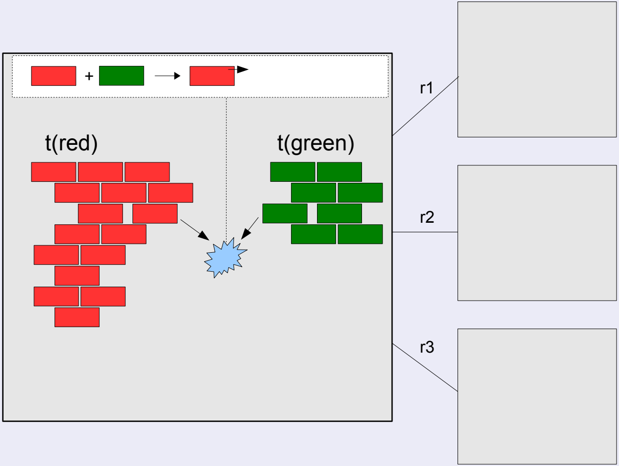
After installing reaction $\mathtt{t(X)} \xrightarrow{0.01} 0$

- We let tuples decade (evaporate like pheromones)
- This is useful to enact time-pertinency
- An agent perceives that the tuple is fading until disappearing
- E.g. `t(s)` represents the temporaneous publication of a service



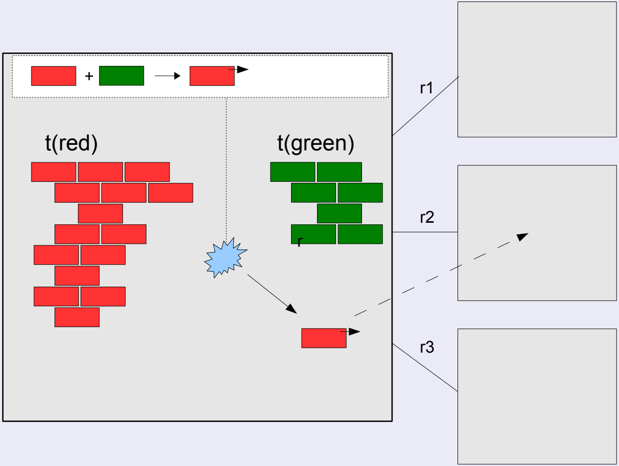
Tuple Transfer

Right-hand side of a reaction can have a firing tuple



From one node to a full biochemical network

Firing tuples are sent to any neighbour, probabilistically a là $S\pi$



On matching and rates

Overcoming discrete matching

- We use first-order terms for tuples and templates
- Matching is by substitution of variables, but it is ranked
- We use an application-dependent match function $\mu(t, t')$
  - yielding 0 is no match, 1 is perfect match, otherwise it is partial match
  - Chemical reactions are applied "modulo match ranking"
  - E.g. with $\mu = 0.5$, actual chemical rate is divided by 2
- A typical scenario of Web-based match-making (i.e. with preferences)

Example of general decay rule: $\mathtt{DECAY} \xrightarrow{r\_dec} 0$

- A specific tuple t decays with chemical rate $\mu(\mathtt{DECAY}, t) * r\_dec$
- E.g., t models a service publication, granted after paying money
- μ inspects how much it was payed, hence tuning service life-time

The calculus

Some credit

$S\pi$-calculus, Ambient, Membrane Computing, stoKLAIM, ProbLinCa

Syntax

$$
\begin{array}{lcll}
t & ::= & \tau\langle n\rangle & \text{Tuple} \\
T & ::= & 0 \mid t \mid t^{\rightsquigarrow} \mid (T \mid T) & \text{Tuple set} \\
L & ::= & [T_i \stackrel{r}{\mapsto} T_o] & \text{Chemical Law} \\
S & ::= & 0 \mid T \mid L \mid (S \mid S) & \text{Space} \\
A & ::= & wait(r) \mid out(\sigma, t) \mid in(\sigma, t) \mid rd(\sigma, t) & \text{Actions} \\
P & ::= & 0 \mid A.P \mid call\ D(\tau_1, \ldots, \tau_n) & \text{Process} \\
C & ::= & 0 \mid [\![S]\!]_\sigma \mid \sigma \stackrel{r}{\rightsquigarrow} \sigma \mid P \mid (C \mid C) & \text{Configuration}
\end{array}
$$

Stochastic Transition System semantics: $C \xrightarrow{\lambda} C'$

- $C \xrightarrow{r} C'$, a CTMC transition with rate r (average duration $1/r$)
- $C \xrightarrow{r\star} C'$, a DTMC immediate transition with likelihood r

The calculus

Semantics

$$
\begin{array}{rcll}
C \mid C' & \xrightarrow{\lambda} & C \mid C'' & \text{if } C' \xrightarrow{\lambda} C'' \\
out(\sigma, \tau\langle n\rangle).P \mid [\![S]\!]_\sigma & \xrightarrow{1\star} & P \mid [\![\tau\langle n\rangle \mid S]\!]_\sigma & \\
rd(\sigma, \tau\langle v\rangle).P \mid [\![\tau'\langle n\rangle \oplus S]\!]_\sigma & \xrightarrow{\mu(\tau,\tau')\star} & P\{\tau/\tau'\}\{v/n\} \mid [\![\tau'\langle n\rangle \mid S]\!]_\sigma & \\
rd(\sigma, \tau\langle n\rangle).P \mid [\![\tau'\langle n+m\rangle \oplus S]\!]_\sigma & \xrightarrow{\frac{n+m}{n}\mu(\tau,\tau')\star} & P\{\tau/\tau'\} \mid [\![\tau'\langle n+m\rangle \oplus S]\!]_\sigma & \\
in(\sigma, \tau\langle v\rangle).P \mid [\![\tau'\langle n\rangle \oplus S]\!]_\sigma & \xrightarrow{\mu(\tau,\tau')\star} & P\{\tau/\tau'\}\{v/n\} \mid [\![S]\!]_\sigma & \\
in(\sigma, \tau\langle n\rangle).P \mid [\![\tau'\langle n+m\rangle \oplus S]\!]_\sigma & \xrightarrow{\frac{n+m}{n}\mu(\tau,\tau')\star} & P\{\tau/\tau'\} \mid [\![\tau'\langle m\rangle \oplus S]\!]_\sigma & \\
wait(r).P & \xrightarrow{r} & P & \\
[\![\tau\langle n+1\rangle^{\rightsquigarrow} \oplus S]\!]_\sigma \mid [\![S']\!]_{\sigma'} \mid \sigma \overset{r}{\rightsquigarrow} \sigma' & \xrightarrow{r(n+1)} & [\![\tau\langle n\rangle^{\rightsquigarrow} \mid S]\!]_\sigma \mid [\![\tau\langle 1\rangle \mid S']\!]_{\sigma'} \mid \sigma \overset{r}{\rightsquigarrow} \sigma' & \\
[\![[T_i \overset{r}{\mapsto} T_o] \mid T \mid S]\!]_\sigma & \xrightarrow{\mu(T_i,T)G(r,T,T|S)} & [\![[T_i \overset{r}{\mapsto} T_o] \mid T_o\{T_i/T\} \mid S]\!]_\sigma &
\end{array}
$$

Gillespie function $G(r, T, S)$

Markovian rate of a reaction with propency r, reactants T, in system S

Some implementation fact

Gillespie "direct" simulation algorithm [Gillespie, 1977]

1. Compute the markovian rate $r_1, \ldots, r_n$ of reactions, let R be the sum
2. Choose one of them probabilistically, and execute its transition
3. Proceed again with (1) after $\frac{1}{R} * ln\frac{1}{\tau}$ seconds, with $\tau = random(0, 1)$

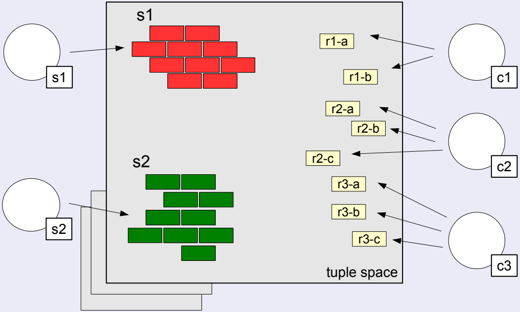
Tuple Space implementation

- Prototyped on top of TuCSoN [Omicini and Denti, 2001]
- Tuple centres programmed with the above algorithm
- The maximum overall rate R should be small enough
- Should otherwise use approximated sim. techniques (τ-leaping)

Outline

The scenario of service ecosystems

Services and requests as tuples



Clients and services as "individuals of an ecology"

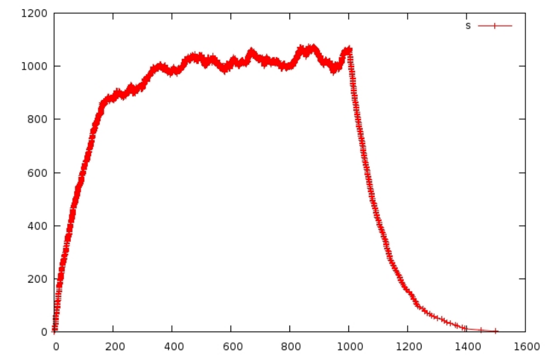
- Unused services fade until completely disappearing
- Concentration of a service increases upon usage
- Similar services compete for survival

Positive-Negative feedback

Idea: Service tuples decay, but can be sustained by a feedback token

- Decay rule: $\mathtt{DECAY} \xrightarrow{r\_dec} 0$
- Feed rule: $\mathtt{publish(SER)} \xrightarrow{r\_feed} \mathtt{publish(SER)} + \mathtt{SER}$

Example simulation: $r\_dec = 0.01, r\_feed = 10$



- time 0: Catalyst Token `publish(S)` is inserted
- time 400: Service `S` reaches an equilibrium
- time 1000: The token is removed (or decays)
- time 1600: Service `S` vanishes

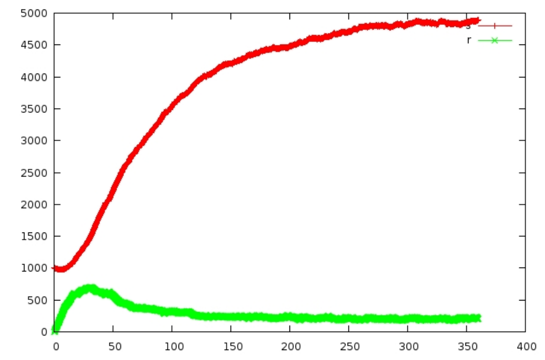
Feedback by using (a.k.a. prey-predator)

Idea: Matching Service-Request sustains the service

- Use rule: $\mathtt{SER} + \mathtt{REQ} \xrightarrow{r\_use} \mathtt{SER} + \mathtt{SER} + \mathtt{toserve}(\mathtt{SER}, \mathtt{REQ})$

Example simulation:

$r\_dec = 0.01$, $r\_use = 0.00005$, $request\_arrival\_rate = 50$



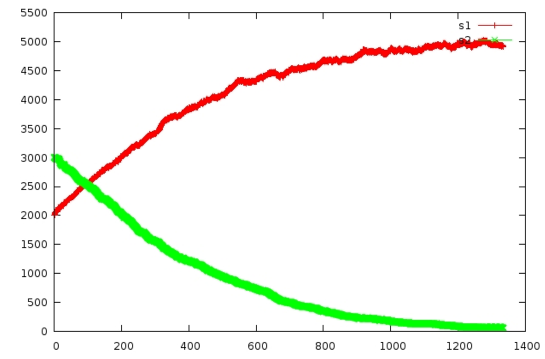
- time 0: Injection of requests raises service level
- time 30: Requests are tamed
- time 350: Unserved requests and service stabilise

Competition

What happens when more services can handle the same requests?

- higher concentration means higher match frequency
- some service may match better the request, being more proper

Example simulation: $r\_use_1 = 0.06, r\_use_2 = 0.04$



- time 0: The two services are in competition for the same requests
- time 100: The one with better use rate (better match) is prevailing
- time 1300: Service *s2* lost competition and fades

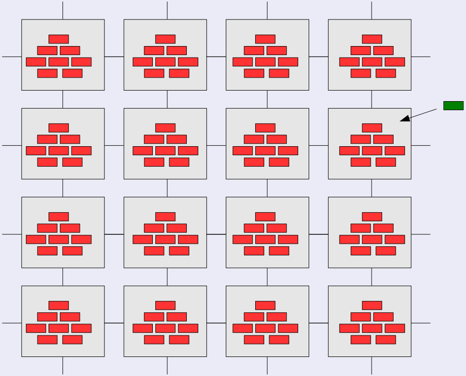
Spatial Diffusion and Competition

One service monopolises a network and its requests

Services continuously diffuse around, by rule:

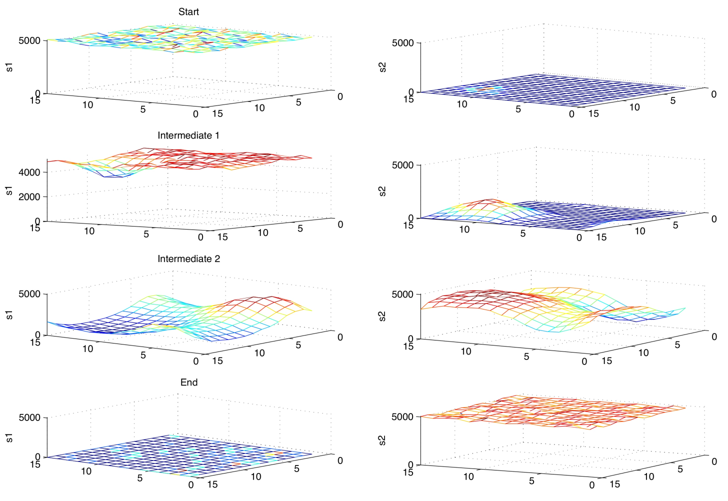
- Diffuse rule: $\mathtt{SER} \xrightarrow{r\_diff} \mathtt{SER}^{\rightsquigarrow}$

Scenario: a better service is injected in a node



Resembling a biological tissue scenario

Example Simulation: $r\_use_1 = 0.05, r\_use_2 = 0.1$



Discussion

Properties

The coordination space achieves the following:

- self-adaptation: the best service is selected over time
- self-optimisation: unused services get disposed
- openness: can deal with incoming new services and requests

Service Composition

Service composition via chemical reaction

Two matching services can compose by rule:

- Compose rule:
  $service(S1) \mid service(S2) \xrightarrow{r\_join} service(compose(S1, S2))$

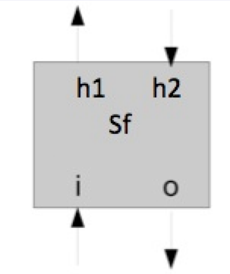
New composite services can be created

- Concentration of composing services is decreased by 1
- Concentration of resulting composite service is increased by 1
- Composite services can be recursively composed with other services

What concrete model for composition?

Concrete Model

Service representation via tuples



Service tuple:

```
service([sf(id502)],
    [in(i),out(o),out(h1),in(h2)],
    [[in(i),out(h1),in(h2),out(o)]]
).
```

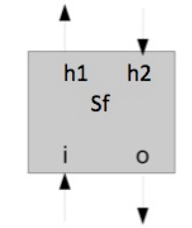
Arguments:

- identifier
- list of input/output ports
- list representing a sequence of ports

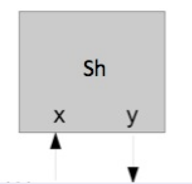
Composing Two Services (1)

Two matching services

One service for flight reservation



One service for hotel booking



Sf and Sh can be composed

- output port h1 matches input port x
- output port y matches input port h2

What chemical rules for composing them?

The Comprehensive Framework for Composition

$$service(C^1, in(B^2) \oplus I, P) \mid service(C'^1, out(B'^2) \oplus I', P')$$
$$\xmapsto{r\_comp}$$
$$service(C \oplus C', link(in(B), out(B')) \oplus I \oplus I', P \oplus P') \quad \texttt{[COMPOSE]}$$

$$service(C, in(B^1) \oplus out(B'^1) \oplus I, P) \xmapsto{r\_link} service(C, link(in(B), out(B')) \oplus I, P) \quad \texttt{[LINK]}$$

$$service(C, I, P) \xmapsto{r\_dec} 0 \quad \texttt{[DECAY]}$$

$$service(C, I, P)^{1} \mid request(ServiceDescription^1, A)$$
$$\xmapsto{r\_use}$$
$$service(C, I, P) \mid session(X^f, A, C, I, P) \quad \texttt{[USE]}$$

$$session(X^f, A, C, I, []) \mid service(C, I, P) \xmapsto{*} service(C, I, P) \mid service(C, I, P) \quad \texttt{[COMPLETE]}$$

$$session(X, A, C, in(B) \oplus I, [in(B)|T] \oplus P) \mid input(X, A, B, M)$$
$$\xmapsto{*}$$
$$session(X, A, C, in(B) \oplus I, T \oplus P) \mid accepted - input(X, A, B, M) \quad \texttt{[INPUT]}$$

$$session(X, A, C, out(B) \oplus I, [out(B)|T] \oplus P) \mid output(X, A, B, M)$$
$$\xmapsto{*}$$
$$session(X, A, C, out(B) \oplus I, T \oplus P) \mid produced - output(X, A, B, M) \quad \texttt{[OUTPUT]}$$

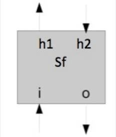
$$session(X, A, C, link(in(B), out(B')) \oplus I, [out(B')|T'] \oplus [in(B)|T] \oplus P) \mid$$
$$produced - output(X, A, B', M)$$
$$\xmapsto{*}$$
$$session(X, A, C, link(in(B), out(B')) \oplus I, T' \oplus T \oplus P) \mid$$
$$accepted - input(X, A, B, M) \quad \texttt{[IN-OUT]}$$

$$session(X, A, C, I, P) \mid abort(X, A) \xmapsto{*} 0 \quad \texttt{[ABORT]}$$

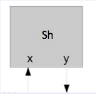
Composing Two Services (2)

From the two matching services . . .

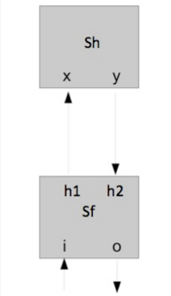
One service for flight reservation



One service for hotel booking



. . . to the composite service




```
service([sf(id502),sh(id103)],
      [in(i),out(o),link(in(x),out(h1)),
       link(in(h2),out(y))],
      [ [in(i),out(h1),in(h2),out(o)],
      [in(x),out(y)] ]
).
```

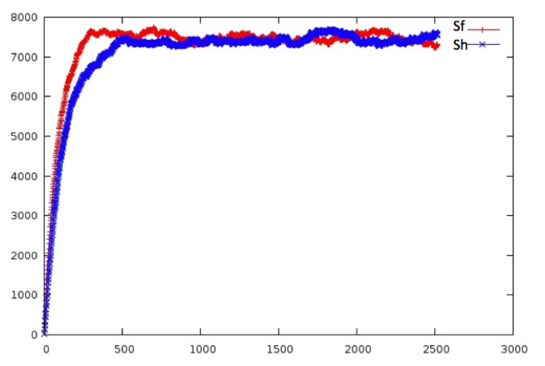
An Example of Service Composition (1)

Scenario

- Two services `sf` and `sh` for flight and hotel booking (respectively)
- Service decay rate $r_{decay} = 0.01$
- Three kinds of requests coming:
  - `rf` (flight reservation) and `rh` (hotel booking) served with $r_{use} = 1.0$
  - `rfh` asking for both a flight and a hotel: served by both `sf` and `sh` with $r_{use} = 0.3$ (partial match!)

An Example of Service Composition (2)

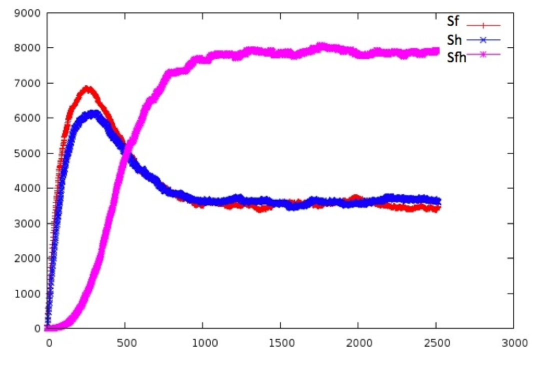
No composition:
$rh\_arrival\_rate = rf\_arrival\_rate = 25$, $rfh\_arrival\_rate = 100$



- The two services reach equilibrium
- Approximate activity level: 7500 per service

An Example of Service Composition (3)

sh and sf compose: composite service sfh in now available



- `sfh` serves only requests `rfh` with $r_{use} = 1.0$
- As a result, now `sfh` competes with `sf` and `sh`

Ongoing Works on Service Composition

Summing up ...

- Exploited the chemical-inspired tuple-space model to ...
- ...devise a model for service composition and competition
- → Prototype implementation in TuCSoN

Work in progress

- Tune reaction rates
- Introduce semantic matching
- Finalize the prototyped implementation
- Find case studies

Outline

Computational Fields

Scenario

- A *spatially distributed network* made of ...
- ... many *computing devices*, usually defined as *nodes*

Computational Fields

- simply put: *a function mapping each node to a value*
- this *value* denotes some relevant aspects of the system state locally to each node

→ A dynamically evolving spatial data structure

Relevance for Pervasive Systems

Two Important Aspects

- Computational fields intrinsically support two important requirements of pervasive systems:
  - *context-awareness*
  - *self-adaptation*

Context-Awareness

- local field value in a node depends on the state of the surrounding nodes

Relevance for Pervasive Systems

Two Important Aspects

- Computational fields intrinsically support two important requirements of pervasive systems:
  - *context-awareness*
  - *self-adaptation*

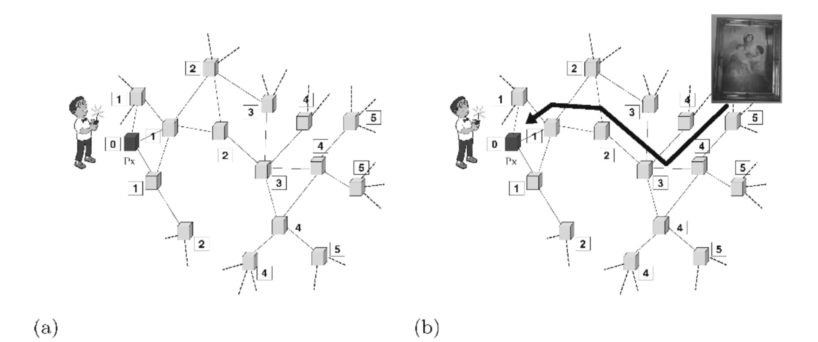
Self-Adaptation

- value mapping occurs on a neighborhood-basis so as to adapt to changes in the network
  - node *failures* and *mobility*, etc.

Uses in Pervasive Domains (I)

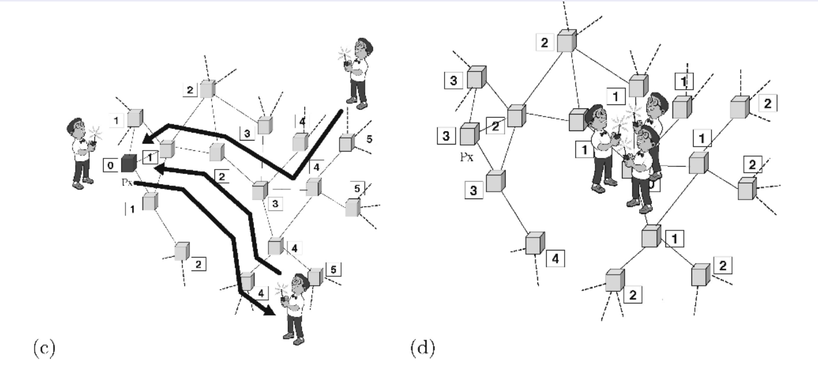
Finding Art Pieces in a Big Museum

From [Mamei and Zambonelli, 2009]



Uses in Pervasive Domains (II)
Discovering in a Big Museum

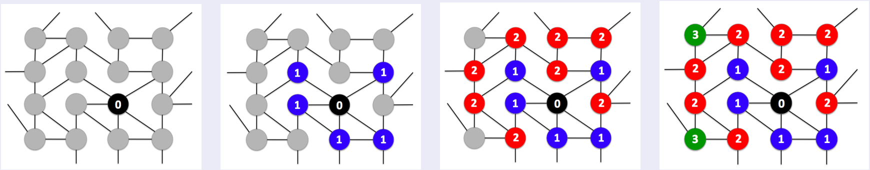
From [Mamei and Zambonelli, 2009]



Examples of Computational Field Algorithms

Gradient

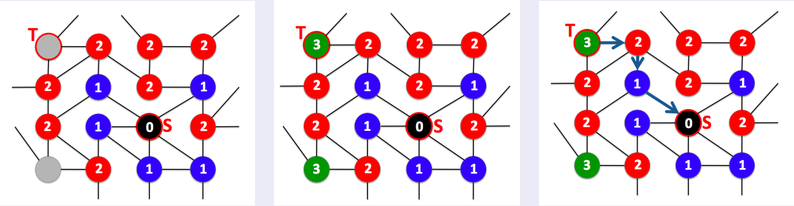
- A computational field where the field value in a specific node depends exclusively on some *notion of distance* from the source node of the gradient
- Example of uses in pervasive systems:
  - find the shortest path to a device in the network
  - build virtual communication channels between devices that need to communicate



Examples of Computational Field Algorithms

Gradient Descent

- A gradient diffusing over the network until reaching a target device (e.g. containing information of interest): the sought information can then follow downhill the gradient until reaching the gradient source
- Example of uses in pervasive systems:
  - data retrieval in spatial settings
  - device discovery



Modelling and Verifying Computational Fields (I)

Our Specification Language

$Spec$	$::=$	$\overline{vdef}\ \overline{rl}$	Specification
rl	$::=$	$\overline{p}$ $--$ Exp $-->$ $\overline{u}$;	Rule
$vdef$	$::=$	X : $[lb..up]$;	Variable Definition
p	$::=$	$c \mid a$	Precondition
c	$::=$	$Exp\ opb\ Exp$	Boolean Condition
a	$::=$	$N := \&neigh[c]$	Neighborhood Assignment
u	$::=$	$V' = Exp$	Update
Exp	$::=$	$Re \mid V \mid Exp\ op\ Exp \mid neigh[Exp]$	Numeric Expressions
opb	$::=$	$>= \mid <= \mid > \mid < \mid = \mid != $	Boolean Operators
op	$::=$	$+ \mid - \mid * \mid /$	Math Operators
$neigh$	$::=$	$any \mid min \mid max$	Neighborhood Functions
V	$::=$	$X \mid N.X \mid @.X$	Variables

Modelling and Verifying Computational Fields (II)

Verification

- Computational Fields modelled via our specification language needs to be verified
- This can be done via *stochastic model checking*

Stochastic Model Checking

- Our Models will be translated into CTMC models, in particular into PRISM models
- This allows to perform *quantitative analysis* related on performance and costs

Modelling Gradient Descent

Model

```
pump   : [0..1];
field  : [0..MAX];
desc   : [0..1];

[]     pump=1 & field>0 -- 1.0 --> field'= 0;

[diff] pump=0           -- 1.0 --> field'= min[@.field]+1;

[move] desc=1 & N:=&any[@.field<field]
                 -- (field-@.field)/@.field/sum((field-@.field)/@.field) -->
       desc'=0 & N.desc'=1;
```

Quantitative Verification: Performance

Time Required to Reach Gradient Source (I)

- As we are in a stochastic domain, this translates to: *which is the probability of reaching source within k time units?*
- This is expressed via the following CSL formula:

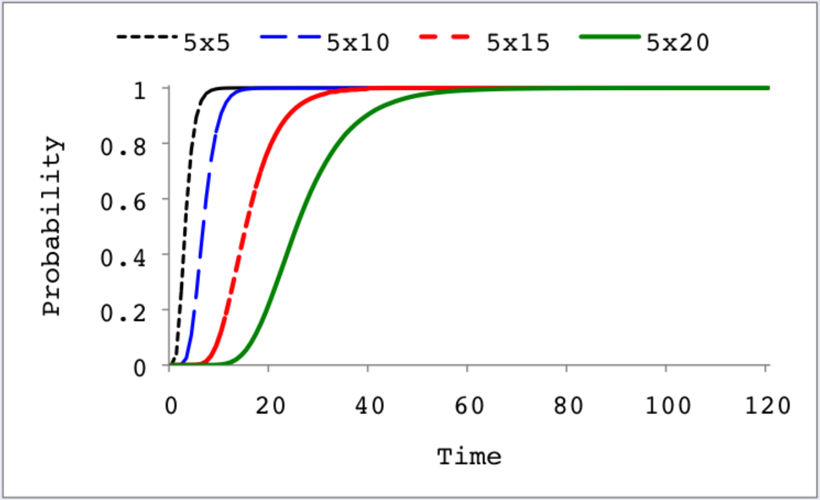
```
P=? [true U<=k "descent_complete"]
```

- Where `descent_complete` is a property specified on the model according to the syntax:

```
property "descent_complete" = exist[ pump=1 & desc=1 ];
```

Quantitative Verification: Performance

Time Required to Reach Gradient Source (II)



Quantitative Verification: Cost

Number of Network Hops to Reach Gradient Source (I)

- As we are in a stochastic domain, this translates to: *which is the expected number of hops necessary to reach source?*
- This is expressed via the following CSL formula:

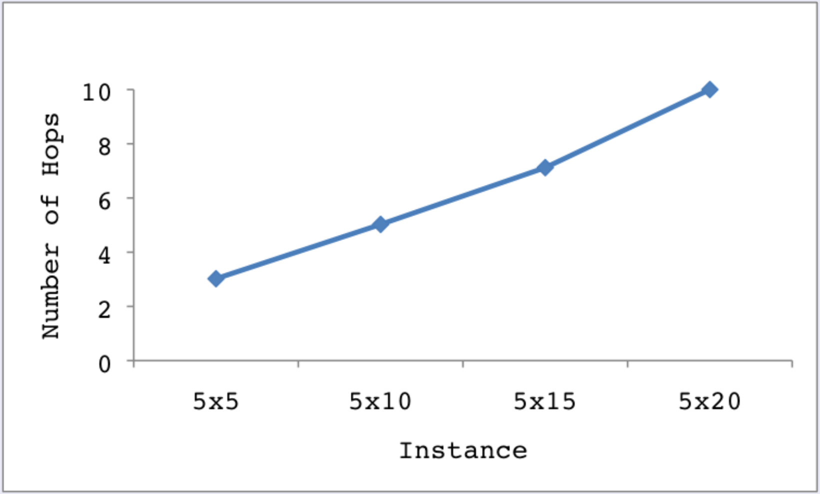
```
R{hops}=? [F "descent_complete"]
```

- Where the `hops` reward structure is specified on the model according to the syntax:

```
rewards "hops" = [move] true : 1;
```

Quantitative Verification: Cost

Number of Network Hops to Reach Gradient Source (II)



Outline

Available Theses on the Presented Topics

Biochemical Tuple Spaces

- Implementation of a a new framework explicitly supporting the biochemical tuple space model

Computational Fields

- Implementation of a framework for modelling and verifying computational fields

Model Checking

- Model checker for approximate model checking starting from simulation tools developed by our research group

Outline

References I

Berryman, A. A. (1992).
The origins and evolution of predator-prey theory.
Ecology, 73(5):1530–1535.

Bonâtre, J.-P. and Le Métayer, D. (1996).
Gamma and the chemical reaction model: Ten years after.
In *Coordination Programming*, pages 3–41. Imperial College Press London, UK.

Bravetti, M., Gorrieri, R., Lucchi, R., and Zavattaro, G. (2004).
Probabilistic and prioritized data retrieval in the linda coordination model.
In Nicola, R. D., Ferrari, G. L., and Meredith, G., editors, *Coordination Models and Languages, 6th International Conference, COORDINATION 2004, Pisa, Italy, February 24-27, 2004, Proceedings*, volume 2949 of *Lecture Notes in Computer Science*, pages 55–70. Springer.

References II

Gillespie, D. T. (1977).
Exact stochastic simulation of coupled chemical reactions.
The Journal of Physical Chemistry, 81(25):2340–2361.

Mamei, M. and Zambonelli, F. (2004).
Programming pervasive and mobile computing applications with the TOTA middleware.
In *Pervasive Computing and Communications, 2004*, pages 263– 273. IEEE.

Mamei, M. and Zambonelli, F. (2009).
Programming pervasive and mobile computing applications: The tota approach.
ACM Trans. Softw. Eng. Methodol., 18(4):1–56.

References III

Menezes, R. and Tolksdorf, R. (2004).
Adaptiveness in linda-based coordination models.
In *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, volume 2977 of *LNAI*, pages 212–232. Springer.

Omicini, A. and Denti, E. (2001).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.

Parunak, H. V. D., Brueckner, S., and Sauter, J. (2002).
Digital pheromone mechanisms for coordination of unmanned vehicles.

In *Autonomous Agents and Multiagent Systems (AAMAS 2002)*, volume 1, pages 449–450. ACM.

References IV

Viroli, M., Casadei, M., and Omicini, A. (2009).
A framework for modelling and implementing self-organising coordination.
In *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1353–1360. ACM.

Zambonelli, F. and Viroli, M. (2008).
Architecture and metaphors for eternally adaptive service ecosystems.
In *IDC'08*, volume 162/2008 of *Studies in Computational Intelligence*, pages 23–32. Springer Berlin / Heidelberg.

Self-Organizing Approaches to System Coordination

Matteo Casadei

Alma Mater Studiorum – Università di Bologna

m.casadei@unibo.it

Progetto di Sistemi Informatici LS

A.Y. 2009/10
June 15, 2010