A Self-Organising Solution to the Collective Sort Problem in Distributed Tuple Spaces

Mirko Viroli  Matteo Casadei  Luca Gardelli

Alma Mater Studiorum – Università di Bologna
{mirko.viroli,m.casadei,luca.gardelli}@unibo.it

ACM SAC 2007
Track on Coordination Models and Languages
Seul - Korea
March 12, 2007
Coordination Models and Languages

Self-Organization and Coordination

The Collective Sort Case

Simulation Framework

Developing a Self-Organizing Solution

Conclusions
What is Coordination?

Settings

- A distributed or concurrent system
- Composed of different entities: agents, processes, components
- Coordination is government of their interactions!

Example models and technologies

- Channels: as in Reo model
- Spaces: as in Linda and all its derivations
  - TuCSoN, Lime, TSpaces, TOTA, KLAIM, …

Even direct communication is a particular case …
A General Meta-Model

Architecture

- Many coordinated entities
- The coordination space hosting *coordination media*
- Coordinated entities interacting with media through primitives
Complex systems, Self-Organization, Emergent Behaviours

Complex systems

- Systems whose dynamics is hardly predictable
- Small changes in initial conditions may lead to completely different behaviours
- They are hard to design: behaviour really emerges without a priori intention

Self-organization as source of complexity

- Designed to adapt to unpredictable changes of the surrounding conditions
- Organization emerges at the global level as a result of local interaction of entities
- A naturally inspired metaphor indeed!
Self-Organization and Coordination

A Reference Scenario

- Should design a coordination space
- Agents require services related to mutual awareness and retrieval of resources
- The system should adapt to dynamism in topology and handle unpredictable agent behaviour and movements

Related Works

- TOTA: co-fields for awareness
- SWARM-LINDA: dynamic movement of tuples in the network
- Many other examples related to stigmergy
The **TOTA** Solution

**TOTA Architecture**

- One tuple space in each node of the dynamic network
- Tuples are:
  - put in the local space by agents
  - **TOTA** spread them in the neighborhood
  - a distributed data structure resembling a *field* is created
  - agents perceive tuples and behave accordingly

**Local/global**

- Interactions are all local, movements are local
- Yet, a global behaviour emerges, in the stigmergy style!
### Some patterns

**Inspired by nature**

- **Diffusion**: some data chunk locally stored, automatically spread around
- **Aggregation**: homogeneous data chunks in the same place are collected
- **Evaporation**: data chunks keep fading until completely vanishing
- **Collective Sorting**: data chunks are moved according to similarity properties
Problem

Definition

Inspired by brood and larvae sorting by ants

- Take a distributed flat set of tuple spaces \((S_1, \ldots, S_n)\)
- Each holding tuples of different kinds \((K_1, \ldots, K_n)\)
- Design a self-organizing solution where:
  - Locally: a tuple can be moved from one space to the other according to local criteria
  - Globally: tuples with same kind are collected in a single space, tuples with different kind are collected in different spaces
Why is this interesting?

Where is Emergence and Adaptiveness?

- The space where a given kind aggregates is uncertain (bifurcation effect)
- Full sorting should be reached independently of initial conditions and ongoing perturbations

Usefulness for Coordination

- If tuples represent information, after a while agents know where to retrieve them
- Supporting tuple space optimization and load balancing
- E.g. having similar tuples in the same place eases consistency checking
An Architecture for the Solution

Elements

- One manager agent for each space $S$ (or possibly more)
- It has the burden of moving tuples away from $S$, at a certain rate
- Decisions taken by relying on a pointwise primitive ($rd$)
- Avoiding global counting operations
- The whole sorting service transparent to user agents
Uniform read

Movement criterion

- How an agent may decide to move a tuple $T$ away from a space $S$?
- The agent should recognise that the kind of $T$ is aggregating more elsewhere..
- We need a new pointwise primitive supporting this reasoning

Uniform read primitive

- $urd(K_1, \ldots, K_n)$
- Reads one tuple belonging to any kind $K_i$, probabilistically!
- The more tuples of kind $K_i$ occur in the space, the more likely one such tuple is read
- This primitive could be implemented e.g. in ReSpecT
The manager agent agenda

Step-by-step behaviour

Consider an agent managing space $S$, and executing this agenda with a fixed rate $r$:

- it draws a tuple kind $K$ of interest, randomly
- it draws a candidate destination tuple space $D$, randomly
- it performs a $urd$ on $S$, obtaining a tuple of kind $K_S$
- it performs a $urd$ on $D$, obtaining a tuple of kind $K_D$
- if $K = K_D \neq K_S$ it moves a tuple of kind $K$ from $S$ to $D$

Intuition

If $K = K_D \neq K_S$ holds, it is likely that $D$ aggregates $K$ more than what $S$ is doing
How we proceed now?

We have an intuition of the strategy, how to design a correct solution?

- Express the design as a formal language
- Execute stochastic simulations, evaluate the results
- If not satisfied, tune the design and proceed again

A pillar work in this direction

- It shows that complex chemical processes (large, discrete systems), can be described by a stochastic approach, rather than by standard differential equations
Stochastic modelling

Start from a stochastic model of a system...

It is basically a transition system \( \langle S, A, \rightarrow \rangle \), where:

- \( S \) is the set of states of the system of interest
- \( A \) is the set of actions (labels for system evolution)
- \( \rightarrow \subseteq S \times A \times \mathbb{R} \times S \) is the transition relation
- (write \( s \xrightarrow{a:r} s' \) for \( \langle s, a, r, s' \rangle \in \rightarrow \))

\( s \xrightarrow{a:r} s' \) means the system may move from \( s \) to \( s' \) by action \( a \) occurring with rate \( r \) (average \( \Delta t \) is \( 1/r \))
Stochastic simulation

General schema

- Start from an initial state
- Choose a new state and a time increase probabilistically
- Proceed and keep track of the evolution history (e.g. to draw a chart)

A simulation step is as follows

- Let $a_1 : r_1, \ldots, a_n : r_n$ be the actions (rates) available in current state
- Draw two random numbers in [0, 1], say $\tau_1$ and $\tau_2$
- Use $\tau_1$ to select an $a_i : r_i$ (probability is $r_i / \sum r_j$)
- Use $\tau_2$ to identify the time increase: $\Delta t = -\ln(1/\tau_2) / \sum r_j$
# A MAUDE library

## What is MAUDE

- It is basically a meta-language for transition systems
- Based on term-rewriting logic
- Can express custom syntax, and rules of transition/rewriting

## A library for stochastic simulations

- The user expresses the transition system $\langle S, A, \rightarrow \rangle$
- The library implements the simulation engine and yields a simulation trace
- The resulting output file is used to chart results

With respect to other simulation frameworks like SPiM and Repast, MAUDE is a general-purpose tool.
A simple example, Sodium-chloride reaction 1/2

**Maude code**

```
op <_,_,_,_> : Nat Nat Nat Nat -> State .
ops ionization deionization : -> Action .
vars Na Na+ Cl Cl- : Nat .

eq < Na,Na+,Cl,Cl- > ==> =
  ( ionization # (float(Na * Cl) * 1.0)}
    -> [< p Na,s Na+,p Cl,s Cl- >] );
  ( deionization # (float(Na+ * Cl-) * 2.0)}
    -> [< s Na,p Na+,s Cl,p Cl- >] ) .
```
A simple example, Sodium-chloride reaction 2/2

Trace

```
<
[300 : < 100,0,100,0 > @ 0.0],
[299 : < 99,1,99,1 > @ 5.2282294378567067e-5],
[298 : < 98,2,98,2 > @ 6.9551290710937174e-5],
[297 : < 97,3,97,3 > @ 8.5491215950091466e-5],
...
[7 : < 61,39,61,39 > @ 3.9845251139158447e-2],
[6 : < 60,40,60,40 > @ 3.9980318990300842e-2],
[5 : < 59,41,59,41 > @ 4.029131950475788e-2],
[4 : < 58,42,58,42 > @ 4.0294167525983679e-2],
[3 : < 57,43,57,43 > @ 4.0424914101137542e-2],
[2 : < 58,42,58,42 > @ 4.0506028901053114e-2],
[1 : < 59,41,59,41 > @ 4.0661029058233995e-2],
[0 : < 60,40,60,40 > @ 4.0695684943167353e-2]
>
```

Chart

![Graph showing Na and Na+ over time]
Modelling Collective Sort in Maude

An initial configuration

\[
\begin{align*}
< 0 & \mathrel{@} (a[100]) \mathrel{|} (b[100]) \mathrel{|} (c[10]) \mathrel{|} (d[10]) > \\
< 1 & \mathrel{@} (a[0]) \mathrel{|} (b[100]) \mathrel{|} (c[10]) \mathrel{|} (d[10]) > \\
< 2 & \mathrel{@} (a[10]) \mathrel{|} (b[50]) \mathrel{|} (c[50]) \mathrel{|} (d[10]) > \\
< 3 & \mathrel{@} (a[50]) \mathrel{|} (b[10]) \mathrel{|} (c[10]) \mathrel{|} (d[50]) > 
\end{align*}
\]

Semantic Rules

- Basically, one for each step of the agent agenda
- The first one creates a new agent (state) at rate \( r \)
- The other rules evolve this state as decisions are taken
- The latter possibly changes the configuration of tuples
Results 1/3

Behaviour in tuple space $S_0$
Results 2/3

Winning tuple in each tuple space

![Graph showing the number of tuples over time for different tuple spaces.](image)
Entropy in each tuple space

Computed as: $\sum -c_K * \ln c_K$ ($c_K$ is the concentration of $K$)
What about convergence?

Local minima for entropy exist!

An example:

- The concentration of tuple a in 0 and 1 is 100% (full aggregation)
- Tuples c and d are never moved away!

This general situation is quite frequent when using complex systems as optimization tools.
The vacuum tuple solution

The problem

- Some tuple spaces might be simply empty (how *urd* works?)
- Not only the relative concentration but also absolute value should be considered
- We need a form of simulated annealing!

A solution

- Each tuple space has also a (fixed) number of vacuum tuples
- If the destination tuple is vacuum, then move the source tuple there!!
Vacuum architecture
A new agenda

Step-by-step behaviour

Consider an agent managing space $S$, and executing this agenda with a fixed rate $r$:

- it draws a tuple kind $K$ of interest, randomly
- it draws a candidate destination tuple space $D$, randomly
- it performs a $urd$ on $S$, obtaining a tuple of kind $K_S$
- it performs a $urd$ on $D$, obtaining a tuple of kind $K_D$
- if $K = K_D \neq K_S$ it moves a tuple of kind $K$ from $S$ to $D$
- if $K \neq K_S$ and $K_D = v$ it moves a tuple of kind $K$ from $S$ to $D$

Intuition

If $K_D = v$ the destination has some emptiness, and hence we move the tuple
New simulations

- Good overall *performance* is achieved when vacuum concentration is 20% of the final number of tuples.
- How can this be designed in advance?
- We need an adaptive mechanism for vacuum!

\[ K = \text{Number of moved tuples} \]
Agent for adaptive vacuum

Step-by-step behaviour

Consider an agent managing space $S$, and executing this agenda with a fixed rate $r$:

- it draws a tuple kind $K$ of interest, randomly
- it draws a candidate destination tuple space $D$, randomly
- it performs a $urd$ on $S$, obtaining a tuple of kind $K_S$
- it performs a $urd$ on $D$, obtaining a tuple of kind $K_D$
- if $K = K_D \neq K_S$ it moves a tuple of kind $K$ from $S$ to $D$
- if $K \neq K_S$ and $K_D = v$ it moves a tuple of kind $K$ from $S$ to $D$
- if $K = K_D \neq K_S$ it drops one vacuum tuple from $S$
- if $K = K_D = K_S$ it adds one vacuum tuple to $S$
New simulations

- The obtained performance is sufficiently far from the *bad* zone
- No significant performance impact on instances that normally converge

\[ K = \text{Number of moved tuples} \]
Conclusions

Experience

- Coordination and Self-Organization
- Provide design-support to adaptive behaviour

Future Work

Putting our simulation framework to test in other contexts
- Cellular automata
- Chemical/Biological modelling
- Towards new computation paradigms