A Case of Self-Organising Environment for MAS: the Collective Sort Problem

Matteo Casadei Luca Gardelli Mirko Viroli

Alma Mater Studiorum – Università di Bologna

{m.casadei,luca.gardelli,mirko.viroli}@unibo.it

EUMAS 2006
Lisbon - Portugal
December 15, 2006

Part I

Introduction

MAS Environment and Self-Organisation

- Due to the complexity of the interactions among agents...
- ...a MAS is a complex system characterised by *unpredictable dynamics and changes*
- How can we coordinate agents' activities?
  - Adopting ideas deriving from environment in self-organising systems...
  - ...hence, using *self-organising techniques* to design MAS environments

Main Challenge

Vision

- How to devise a correct and appropriate design of a self-organising environment for MAS?
    - Using *simulation tools* in the analysis and the design stage
- Exploit, to this end, a framework for simulations of complex systems

Main Challenge

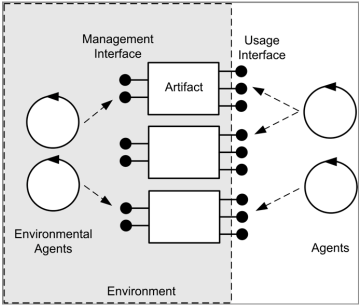
Vision

- How to devise a correct and appropriate design of a self-organising environment for MAS?
  - Using *simulation tools* in the analysis and the design stage
- Exploit, to this end, a framework for simulations of complex systems

Objective of This Paper

Study a particular problem deriving from self-organisation, known as Collective Sort, applying it to the engineering of MAS environments

The Agents & Artifacts (A&A) Meta-Model



- Artifacts encapsulate *resources and services* provided by the environment to agents
- Two different kinds of agent:
  - *environmental* agents *manage* artifacts, that is, they *regulate* the behaviour of artifacts
  - *user* agents *exploit* artifacts, in order to coordinate their activities and to achieve individual and social goals

Part II

From the Paper

Scenario

- We have a multiagent system:
  - its environment has items of different kind and ...
  - ... *Environmental agents*, that have to order items on the basis of a common criterion

Scenario

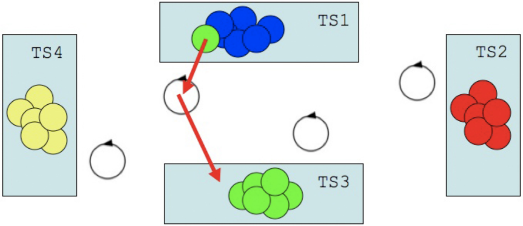
- We have a multiagent system:
  - its environment has items of different kind and ...
  - ... *Environmental agents*, that have to order items on the basis of a common criterion

Why is Collective Sort interesting?

- Environmental agents order informations available in the artifacts ...
- ... hence, user agents who exploit an environment based on Collective Sort, can easily find the informations they need
  - Environmental agents (together with artifacts) provide a service for user agents!

A Possible Architecture

- We have:
  - a set of sorter agents (*environmental agents*) managing . . .
  - . . . tuple spaces (*artifacts*)



- Each *sorter agent* can read, insert and remove tuples from tuple spaces
- The *emergent property* is to achieve complete order of tuple spaces

Agenda of Sorter Agents

1. He chooses a destination tuple space D (different from the source one S)
2. He chooses a kind K of tuple
3. He reads a tuple Ta from S
4. He reads a tuple Tb from D
5. He moves a tuple of kind K from S to D if *kind of* Tb $=$ K and *kind of* Ta$\neq$K

Agenda of Sorter Agents

1. He chooses a destination tuple space D (different from the source one S)
2. He chooses a kind K of tuple
3. He reads a tuple Ta from S
4. He reads a tuple Tb from D
5. He moves a tuple of kind K from S to D if *kind of* Tb = K and *kind of* Ta≠K

Goal of the Agenda

This agenda has a very simple goal:

- avoiding to have tuples of a certain kind in tuple spaces that are not attractor for that kind

Uniform Reading

A New Primitive

The previous readings are performed by the uniform read (`urd`) primitive: given some tuple templates (`a(X)`, `b(X)`), a tuple is read in a probabilistic way among all the matching tuples

Uniform Reading

Example

If a tuple space contains:

- 10 tuples `a(1)`
- 3 tuples `b(1)`
- 17 tuples `b(2)`

performing the uniform reading `urd( a(X), b(Y) )`, we have a probability of

- 66% to read a tuple matching `b(Y)`
- 33% to read a tuple matching `a(X)`

E.g.: if `urd` returns a tuple matching `b(X)`, nothing is said about the probability of reading `b(1)` or `b(2)`: the choice is *non-deterministic*

How to Simulate Collective Sort

In order to *model* and *simulate* the scenario depicted for Collective Sort, we used a *framework for stochastic simulations* we have developed

How to Simulate Collective Sort

In order to *model* and *simulate* the scenario depicted for Collective Sort, we used a *framework for stochastic simulations* we have developed

Stochasticity

- Stochasticity is usually adopted to model and analyse complex systems, because:
  - it can cope with *non-determinism*, that is proper to complex systems
  - in fact, *stochastic models* can represent systems that have an aleatory time evolution, modelling *time* by an *aleatory* variable

The Stochastic Simulation Framework (I)

- It was implemented using MAUDE

The Stochastic Simulation Framework (I)

- It was implemented using MAUDE

What is MAUDE?

- It is a programming language used to model and specify a wide range of systems
- It is based on *equational* and *rewriting logic*

The Stochastic Simulation Framework (II)

- It is based on a simple idea deriving from stochastic pi-calculus
- The idea is to model a stochastic system by a *labelled transition system*

The Stochastic Simulation Framework (II)

- It is based on a simple idea deriving from stochastic pi-calculus
- The idea is to model a stochastic system by a *labelled transition system*

Transitions

- Every transition is of the kind:

$$S \xrightarrow{r:a} S'$$

- A system can move from state S to state S' by an action a with a given rate r

Understanding the Framework

- A very simple example: the *Na-Cl* chemical reaction dynamics

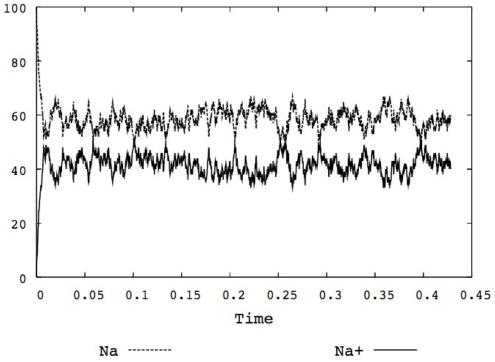
```
op <_,_,_,_> : Nat Nat Nat Nat -> State .

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 Simulation 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]
>
```

From Traces to Charts

- Easy off-line generation of charts starting from traces!

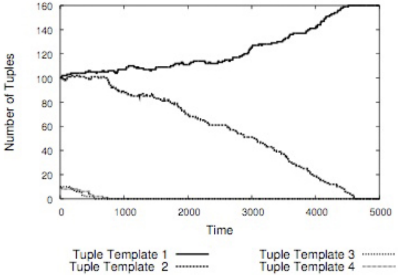


Simulation: a Test Instance

We simulated a Collective Sort instance with 4 tuple spaces and 4 tuple kinds

```
op SS : -> State .
  eq SS = ( init | < 0 @ ('a[ 100] ) | ('b[ 100] ) | ('c[ 10] ) | ('d[ 10] )> |
                   < 1 @ ('a[   0] ) | ('b[ 100] ) | ('c[ 10] ) | ('d[ 10] )> |
                   < 2 @ ('a[  10] ) | ('b[  50] ) | ('c[ 50] ) | ('d[ 10] )> |
                   < 3 @ ('a[  50] ) | ('b[  10] ) | ('c[ 10] ) | ('d[ 50] )  .
```

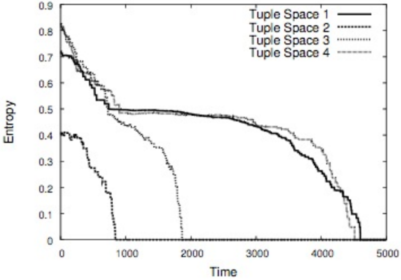
Simulation Results (I)



- Trend of a tuple space: only tuples of one kind aggregate here
- Tuples of different kinds disappear: they are moved to other tuple spaces

Simulation Results (II)

Entropy is a measure of the *chaos* in a tuple space



Entropy reaches *zero* in every tuple space:
Complete order!

Some Observations

Limits of this approach

- We discovered that, in some cases, the adopted solution fails to reach *complete order*
- Indeed, there are certain states attracting the evolution of the Collective Sort and characterized by a positive entropy value, e.g.:

```
< 0 @ ('a[ 20 ] ) | ('b[   0] ) | ('c[  0] ) | ('d[  0] ) > |
< 1 @ ('a[ 140] ) | ('b[   0] ) | ('c[  0] ) | ('d[  0] ) > |
< 2 @ ('a[   0] ) | ('b[ 260] ) | ('c[  0] ) | ('d[  0] ) > |
< 3 @ ('a[   0] ) | ('b[   0] ) | ('c[ 80] ) | ('d[ 80] ) >
```

We need a new approach in order to guarantee *complete order* whatever instance we are going to use!

Part III

Ongoing/(Future) Work

Modelling the Vacuum (I)

- In the Brood Sorting, ants pick a brood up and release it in a place with a higher *concentration* of brood
- Concentration is expressed over units of space: hence, an ant compares the amount of brood with vacuum in a unit of space

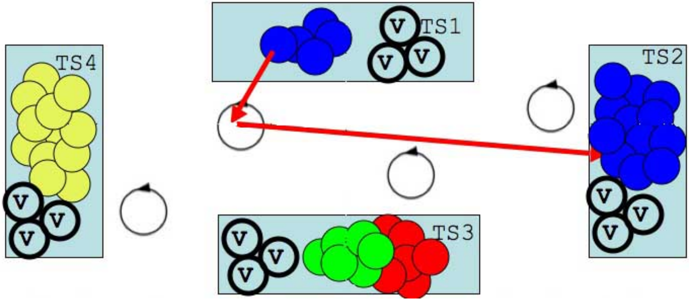
Modelling the Vacuum (I)

- In the Brood Sorting, ants pick a brood up and release it in a place with a higher *concentration* of brood
- Concentration is expressed over units of space: hence, an ant compares the amount of brood with vacuum in a unit of space

We decided to model vacuum in our Collective Sort example using a new kind of tuple: the vacuum tuple

- Each tuple space has the same amount of vacuum tuples, fixed since the beginning (static amount)
- The `urd` primitive can now yield a tuple of kind vacuum

Modelling the Vacuum (II)



- The concentration of tuples becomes relative to vacuum!
  - TS1 has less blue tuples than TS2, because the latter occupies less vacuum
  - TS3 aggregates green and red tuples: now, they can fill vacuum in other tuple spaces!

The New Agenda of Sorter Agents

1. He chooses a destination tuple space D (different from the source one S)
2. He chooses a kind K of tuple
3. He reads a tuple Ta from S
4. He reads a tuple Tb from D
5. He moves a tuple of kind K from S to D if *kind of* Tb = K and *kind of* Ta≠K

A new Rule

He moves a tuple of kind K from S to D if *kind of* Tb = vacuum and *kind of* Ta≠ K

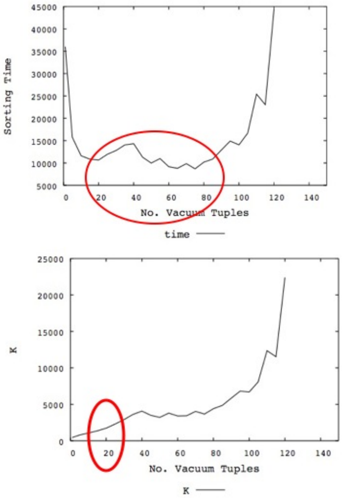
The New Test Instance

A new Collective Sort instance, characterized by 2 tuple space aggregating the same quantity of tuples [a]

```
< 0 @ ('a[ 50] ) | ('b[   0] ) | ('c[  0] ) | ('d[   0] ) > |
< 1 @ ('a[ 50] ) | ('b[   0] ) | ('c[  0] ) | ('d[   0] ) > |
< 2 @ ('a[  0] ) | ('b[ 100] ) | ('c[  0] ) | ('d[   0] ) > |
< 3 @ ('a[  0] ) | ('b[   0] ) | ('c[ 100] )| ('d[ 100] ) >
```

We simulated this instance with different concentrations of vacuum tuples

Results



K = Number of moved tuples

- Best *total sorting time* if we use a vacuum concentration of [20% - 80%] of the final number of tuples expected in each tuple space
- Good *performance* to *cost ratio* with a vacuum concentration of 20% of the final number of tuples expected in each tuple space

Self Adapting Vacuum: General Idea

Observation

The previous results show the performance of the strategy with different concentrations of vacuum tuple:

- In some case, we don't know a priori the concentration of tuples!
- How to devise a solution without any need to choose an initial static concentration of vacuum?

Self Adapting Vacuum: General Idea

Solution

- We need a *truly self-organising solution*!
- We devised a strategy with a initially very low concentration of vacuum:
  - sorter agents locally increase vacuum if tuple spaces reach a local minimum
  - sorter agents locally decrease vacuum when tuple spaces tend to complete order

The Extended Agenda of Sorter Agents

1. He chooses a destination tuple space D (different from the source one S)
2. He chooses a kind K of tuple
3. He reads a tuple Ta from S
4. He reads a tuple Tb from D
5. He moves a tuple of kind K from S to D if *kind of* Tb = K and *kind of* Ta≠K
6. He moves a tuple of kind K from S to D if *kind of*Tb = vacuum and Ta ≠ K

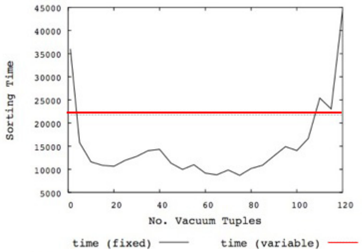
The Extended Agenda of Sorter Agents

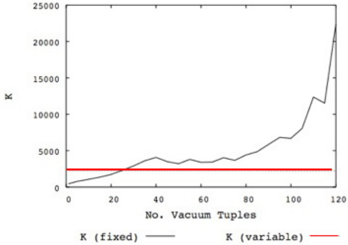
1. He chooses a destination tuple space D (different from the source one S)
2. He chooses a kind K of tuple
3. He reads a tuple Ta from S
4. He reads a tuple Tb from D
5. He moves a tuple of kind K from S to D if *kind of* Tb $=$ K and *kind of* Ta$\neq$K
6. He moves a tuple of kind K from S to D if *kind of*Tb $=$ vacuum and Ta $\neq$ K

Two New Rules

1. if K $=$ Tb $\neq$ Ta then He **drops** one vacuum tuple from tuple space S
2. if K $=$ Tb $=$ Ta then He **adds** one vacuum tuple from tuple space S

Simulation and Results





K = Number of moved tuples

- The self-adapting solution presents a value both for *total convergence time* and *number of moved tuples* that are the mean of the values obtained for the different vacuum concentrations of the previous strategy
- These values are far from bad performance zone
- No significant performance impact on instances that normally converge (like the initial simulated instance)

Conclusions and Future Work

Conclusions

- We exploited our *stochastic simulation framework* as a suitable tool to devise self-organising solutions in the engineering of MAS environments
- In particular, in this work, we concentrated on the *Collective Sort* problem
- By means of *stochastic* simulations, we found a good strategy to solve the *problem of convergence*

Conclusions and Future Work

Conclusions

- We exploited our *stochastic simulation framework* as a suitable tool to devise self-organising solutions in the engineering of MAS environments
- In particular, in this work, we concentrated on the *Collective Sort* problem
- By means of *stochastic* simulations, we found a good strategy to solve the *problem of convergence*

Future Work

- Improve the simulation framework
- Apply the simulation framework to other scenarios of self-organising environments for MAS
- Better exploration of the Collective Sort problem

Thank you!

Thank you!

Questions?