

Self-Organising Approaches to Coordination

Second Faculty of Engineering
Informatics System Design LS

Matteo Casadei¹ and **Mirko Viroli¹**

¹Alma Mater Studiorum – Università di Bologna
{m.casadei,mirko.viroli}@unibo.it

Cesena - March 6, 2008

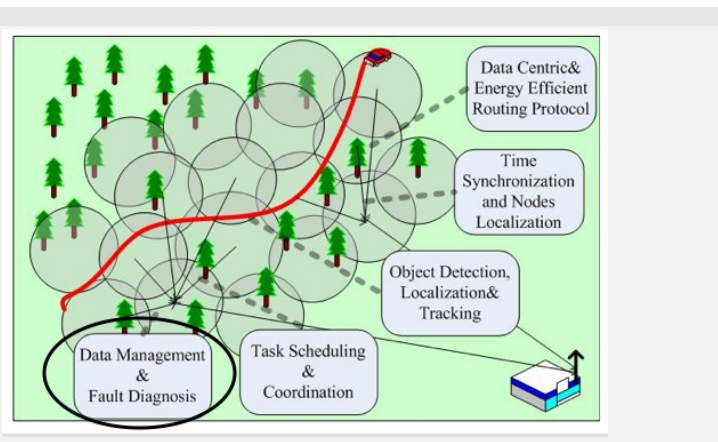


The Need for Distributing Information

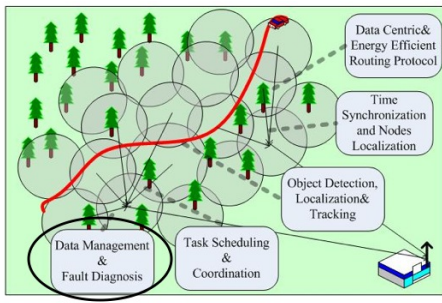
- Today's software systems generate tons of informations
- Some examples are:
 - Wireless sensor networks
 - Pervasive systems



Information Distribution: Wireless Sensor Networks (1)



Information Distribution: Wireless Sensor Networks (2)

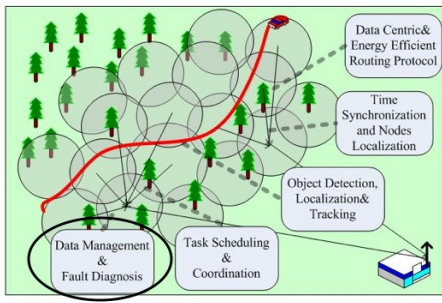


Considerations

- Tons of data is generated!
- Usually some nodes act as sink nodes so as to gather all the data generated that is then sent to a base station
- Accordingly, *routing* becomes a key issue of the design of such networks, in particular w.r.t. energy consumption



Information Distribution: Wireless Sensor Networks (3)



Current Limits

- Sensors are called motes and though having computational capabilities, they have a very limited amount of memory:
 - Impossible to keep data distributed in the network
 - Sink nodes and routing algorithms are necessary
- Reducing energy consumption is a big challenge that affects even routing



Wireless Sensor Networks: Opportunities

Future Technological Steps (Hopefully!)

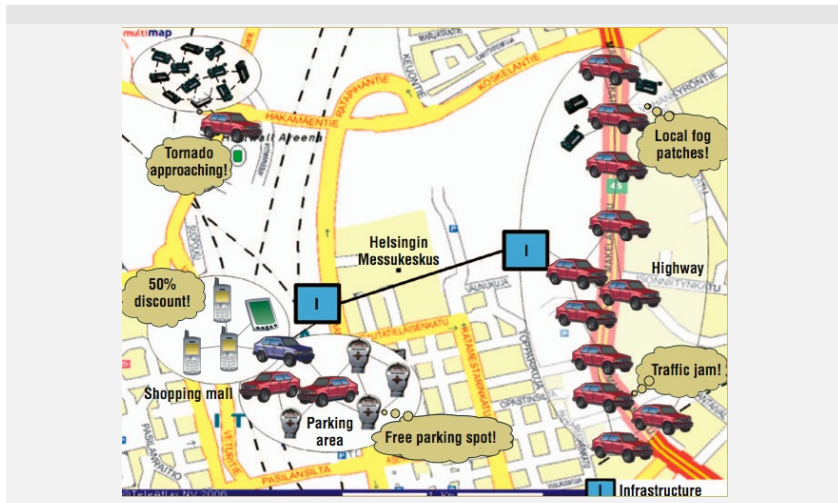
- Motes with larger memories
- Long-lasting batteries

Emerging Opportunities

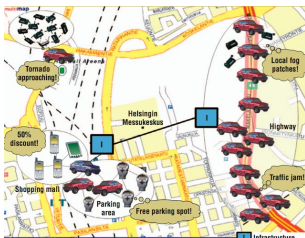
- Keeping the data distributed in the network could improve data access from the standpoint of:
 - Energy saving
 - Wireless sensor network having a more uniform set of motes:
 - Sink nodes would be no longer required so as to make routing algorithms less important



Information Distribution: Pervasive Systems (1)



Information Distribution: Pervasive Systems (2)

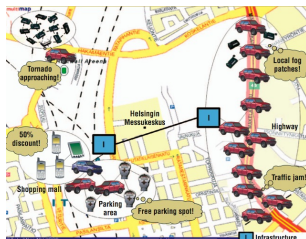


Scenario

- Different kinds of networks (wireless sensors and mobile networks) featuring embedded and mobile devices (such as smart phones, pda, etc.)
- People centric platforms:
 - A pervasive system uses data generated by sensors so as to coordinate different activities, which provides users (citizens) with a new life experience
- To this end, data needs to be collected, organized and shared in an effective way!



Information Distribution: Pervasive Systems (2)



Issues

- Coordination among processes (agents) that act at the software level of the pervasive system is essential for providing people with real-time information
- As a consequence, the way data perceived from the environment are treated and organized becomes key to address the above issue



Review

- Information distribution becomes key for coordination among processes of nowadays software systems
- this talk focuses on information management from the standpoint of tuple-space systems
- the way tuples are organized and distributed on a network of distributed tuple spaces requires the solution of complex problems . . .
- . . . in particular from the viewpoint of interaction!

Formal methods have proven to be crucial for the design of solutions to such a kind of problems!



Importance of Formal Methods

- Today's software systems feature *lots of components* interacting with:
 - High levels of *distribution* and *concurrency*
- The complexity of such systems has continuously been increasing in last years



Complexity (1)

- Complexity viewed from two standpoints
 1. **Complexity of the interaction among systems' components**
 2. Complexity of representing today's software systems



Coping with Interaction Complexity (1)

Scenario

- Interaction complexity leads to systems whose behavior is intrinsically *unpredictable*
- The theory of *complex systems* has recently been exploited to study such systems from the standpoint of *coordination* among system's components

Challenge

- Finding principles and methodologies useful for engineering such systems
 - *Self-organization* has been successfully applied to engineering such systems
- Great need for a formal framework to help using self-organization with engineering complex systems



Coping with Interaction Complexity (2)

Challenge

- Finding principles and methodologies useful for engineering such systems
 - *Self-organization* has been successfully applied to engineering of such systems
- Great need for a formal framework to help using self-organization with the engineering complex systems

A Formal Framework for Stochastic Simulations

- Developed a framework for:
 - formally representing complex systems and ...
 - ...stochastically simulating their behavior
- This makes it possible to apply self-organization techniques to the engineering of complex systems even in the **early stages of the software development processes!**



A Design Methodology for Complex Systems

How to Design a Complex System

- **Modeling:** express the design as a formal language
- **Simulation:** execute stochastic simulations, evaluate the results
- **Tuning:** if not satisfied, tune the design and proceed again

A Pillar Work in This Direction

- D.Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions", 1977
- 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 the 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 Δt 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

- Let $a_1 : r_1, \dots, a_n : r_n$ be the actions (rates) available in current state
- Draw two random numbers in $[0, 1]$, say τ_1 and τ_2
- Use τ_1 to select an $a_i : r_i$ (probability is $r_i / \sum r_j$)
- Use τ_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-Chlorine 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-Chlorine Reaction 2/2

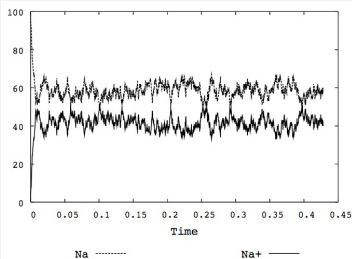
Trace

```

<
[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



Problems Solved by Using the MAUDE-Based Stochastic Framework

- As a way to apply self-organization to the engineering of complex systems, our framework has been tested on two case studies
 - Collective Sort
 - SwarmLinda
- By using the framework, we found a self-organizing solution both to Collective Sort and SwarmLinda



Collective Sort: Introduction

Problem Definition

Inspired by brood and larvae sorting by ants

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



Why Is Collective Sort Interesting?

Where are Emergence and Adaptiveness?

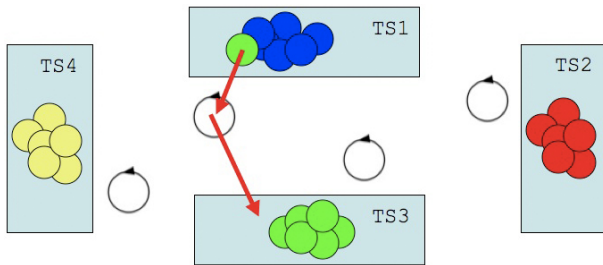
- The space where a given kind aggregates is uncertain (bifurcation effect)
- Full sorting should be reached independently of initial conditions and ongoing perturbations

Usefulness for Coordination

- Since 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 sorter agent for each space S (or possibly more)
- It has the task 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

- $urd(K_1, \dots, K_n)$
- Reads one tuple belonging to any kind K_i , probabilistically!
- The more tuples of kind K_i occur in the space, the more readings of such tuples occur



The Sorter-Agent Agenda

Step-by-step behaviour

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

1. it draws a candidate destination tuple space D , randomly
2. it performs a *urd* on S , obtaining a tuple of kind K_S
3. it performs a *urd* on D , obtaining a tuple of kind K_D
4. if $K_D \neq K_S$, it moves a tuple of kind K_D (if any in S) from S to D

Intuition

If $K_D \neq K_S$ holds, the transfer of a K_D tuple from S to D is likely to make S (D) a stronger aggregator of K_S (K_D) tuples



Modelling Collective Sort in MAUDE

An Initial (Chaotic) Configuration

```
< 0 @ (K1 [25]) | (K2 [25]) | (K3 [25]) | (K4 [25]) > |  
< 1 @ (K1 [25]) | (K2 [25]) | (K3 [25]) | (K4 [25]) > |  
< 2 @ (K1 [25]) | (K2 [25]) | (K3 [25]) | (K4 [25]) > |  
< 3 @ (K1 [25]) | (K2 [25]) | (K3 [25]) | (K4 [25]) >
```

Semantic Rules

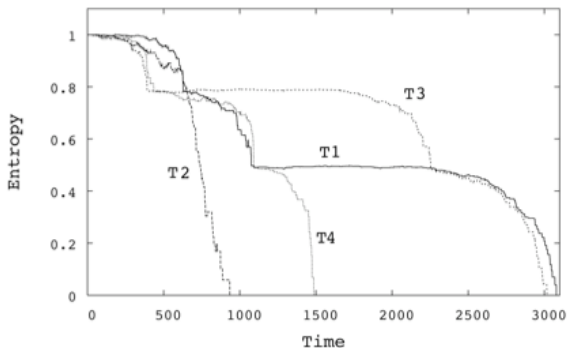
- Basically, one for each step of the agent agenda
- The first one creates a new agent (state) at rate r
- The other rules evolve this state as decisions are taken
- The latter possibly changes the configuration of tuples



Results

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:

```

< 0 @ (K1[100]) | (K2[0]) | (K3[0]) | (K4[0]) > |
< 1 @ (K1[0]) | (K2[69]) | (K3[0]) | (K4[0]) > |
< 2 @ (K1[0]) | (K2[31]) | (K3[0]) | (K4[0]) > |
< 3 @ (K1[0]) | (K2[0]) | (K3[100]) | (K4[100]) >
  
```

- The concentration of tuple K2 in 1 and 2 is 100% (full aggregation)
- Tuples K3 and K4 are never moved away!
- Worse, neither 1 nor 2 are going to become aggregators for all the K2 tuples in the system

With 4 tuple spaces, 5% of the simulations end in a local-minimum state (15% with 7 tuple spaces)



The Noise-Tuple Solution (1)

Problem

- The current sorter-agent agenda does not avoid the case where 2 tuple spaces aggregate the same tuple kind
- **Nothing is done when** $K_S = K_D$, so ...
- ... the aggregation of the same kind on two different tuple spaces leads to the aggregation of two other kinds in a single tuple space
- We need a form of *simulated annealing*!

A Solution

- Introduction of a new kind called `noise`!



The Noise-Tuple Solution (2)

Noise: Meaning

- Noise can be regarded as a perturbation, a probabilistic alteration of agent-action correctness
- Since a urd can now yield a noise tuple, we can make K_2 tuples leave 2 and aggregate in 1

Choosing a Strategy for Managing Noise Concentration

- We need a fully-adaptive noise strategy:
 - Start with an initially very-low noise concentration (for each tuple space)
 - Increase the concentration when the state approaches a local minimum
 - Decrease the concentration when the state goes toward an increasingly ordered state



The Noise Tuple Solution

Choosing a Strategy for Managing Noise Concentration

- Start with an initially very-low noise concentration (for each tuple space)
- Increase the concentration when the state approaches a local minimum
- Decrease the concentration when the state goes toward an increasingly ordered state

A Concrete Solution to Noise Concentration

- Initially, every tuple space features only one noise tuple
- noise is increased every time two tuple spaces appear to be aggregator of the same kind
- noise is decreased every time a tuple is correctly transferred with no noise involved



A New Agenda

Step-by-Step Behaviour

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

1. It draws a candidate destination space D , randomly
2. It performs a *urd* on S , obtaining a tuple of kind K_S
3. It performs a *urd* on D , obtaining a tuple of kind K_D
4. If $noise = K_S \neq K_D \neq noise$, it moves a tuple of kind K_D (if any in S) from S to D
5. If $noise \neq K_S \neq K_D = noise$, a tuple of kind K_S is moved from S to D
6. If $noise \neq K_D = K_S$, noise is increased by 1 in S
7. if $noise \neq K_S \neq K_D \neq noise$, noise is decreased by 1 in S (a tuple K_D is moved from S to D)



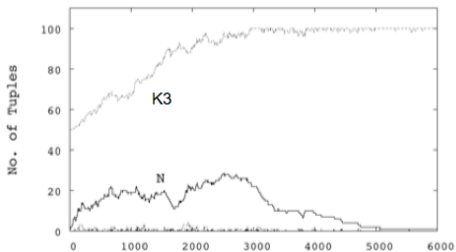
New Simulation: a Worst-Case Symmetric Local Minimum

An example:

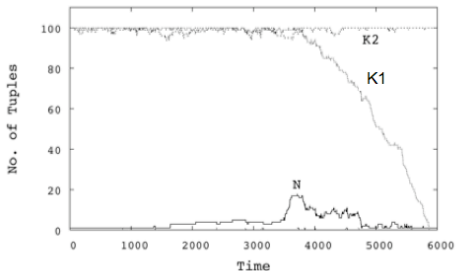
```
< 1 @ (K1[100]) | (K2[100]) | (K3[0]) | (K4[0]) > |  
< 2 @ (K1[0]) | (K2[0]) | (K3[50]) | (K4[0]) > |  
< 3 @ (K1[0]) | (K2[0]) | (K3[50]) | (K4[0]) > |  
< 4 @ (K1[0]) | (K2[0]) | (K3[0]) | (K4[100]) > |
```



Results



- Winning tuple in Tuple space 2

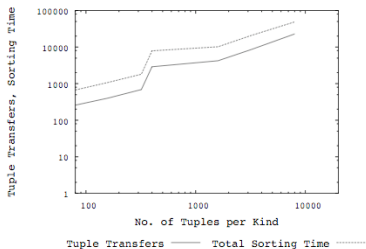


- Winning tuple in Tuple space 1

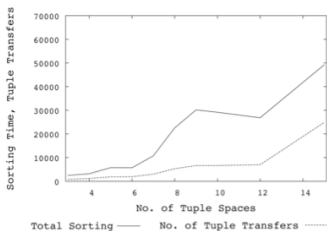
N = noise-tuple concentration



Performance (1): Scalability



- Scalability for different concentrations of tuples ($Num_{TS} = 4$)

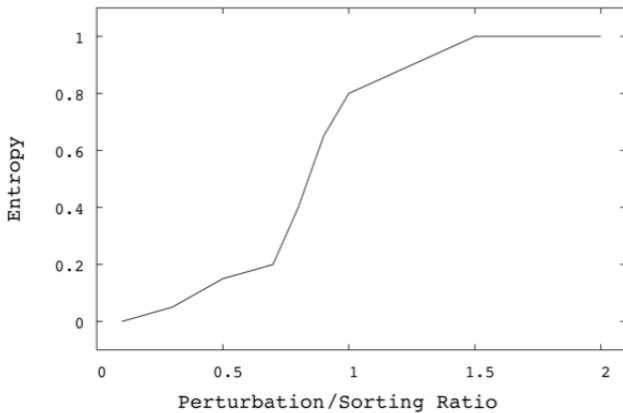


- Scalability for different numbers of tuple spaces ($Num_{tuples} = 400$ per kind)

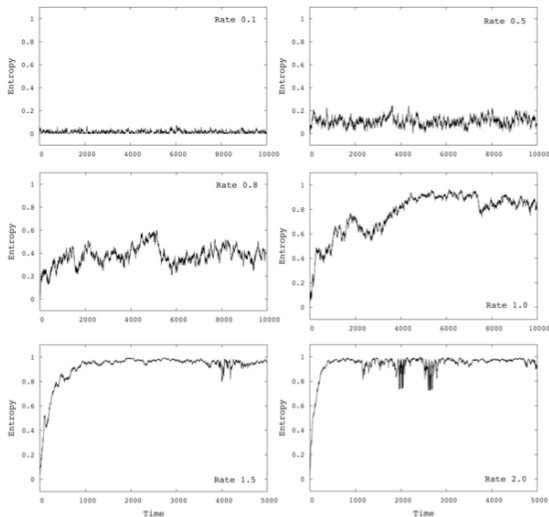
The new strategy guarantees that full sorting is reached ever!



Performance (1): Reactiveness (a)



Performance (1): Reactiveness (b)



Other Simulated Configurations

| i | T1 | T2 | T3 | T4 |
|-----|-----|----|----|----|
| K1 | 100 | 0 | 0 | 0 |
| K2 | 100 | 0 | 0 | 0 |
| K3 | 100 | 0 | 0 | 0 |
| K4 | 100 | 0 | 0 | 0 |



| f | T1 | T2 | T3 | T4 |
|-----|-----|-----|-----|-----|
| K1 | 0 | 0 | 0 | 100 |
| K2 | 100 | 0 | 0 | 0 |
| K3 | 0 | 0 | 100 | 0 |
| K4 | 0 | 100 | 0 | 0 |

| i | T1 | T2 | T3 | T4 |
|-----|----|----|----|----|
| K1 | 50 | 50 | 0 | 0 |
| K2 | 50 | 50 | 0 | 0 |
| K3 | 0 | 0 | 50 | 50 |
| K4 | 0 | 0 | 50 | 50 |

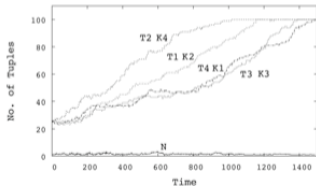
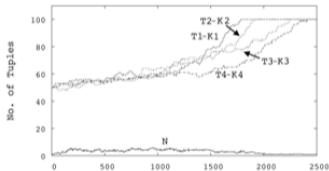
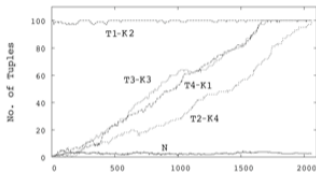


| f | T1 | T2 | T3 | T4 |
|-----|-----|-----|-----|-----|
| K1 | 100 | 0 | 0 | 0 |
| K2 | 0 | 100 | 0 | 0 |
| K3 | 0 | 0 | 100 | 0 |
| K4 | 0 | 0 | 0 | 100 |

| i | T1 | T2 | T3 | T4 |
|-----|----|----|----|----|
| K1 | 25 | 25 | 25 | 25 |
| K2 | 25 | 25 | 25 | 25 |
| K3 | 25 | 25 | 25 | 25 |
| K4 | 25 | 25 | 25 | 25 |



| f | T1 | T2 | T3 | T4 |
|-----|-----|-----|-----|-----|
| K1 | 0 | 0 | 0 | 100 |
| K2 | 100 | 0 | 0 | 0 |
| K3 | 0 | 0 | 100 | 0 |
| K4 | 0 | 100 | 0 | 0 |



Conclusion

Experience

- Self-Organization applied to coordination
- Provided design-support to adaptive behaviour
- The shown solution is described in detail in a paper invited to *Science of Computer Programming*

Future Work

Putting our simulation framework to test in other contexts

- Cellular automata
- Chemical/Biological modelling
- Towards new computation paradigms

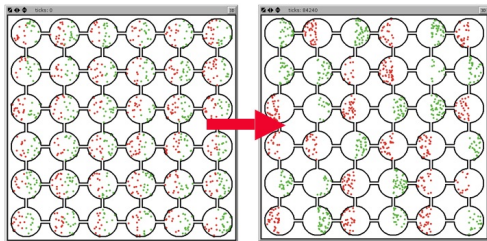


Current Work (1)

General-Topology Networks

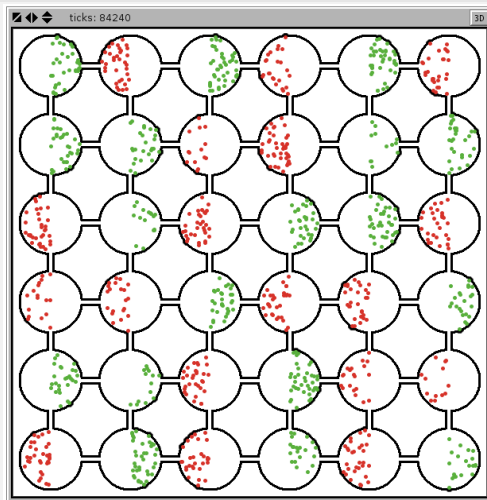
- We applied the original algorithm to general-topology networks
- Example:
 - Grid-topology network

Grid Topology



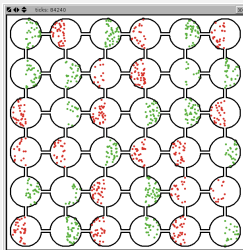
Current Work (2)

Grid Topology: Results



Current Work (2)

Grid Topology: Results



Local emergence of clustering!!!

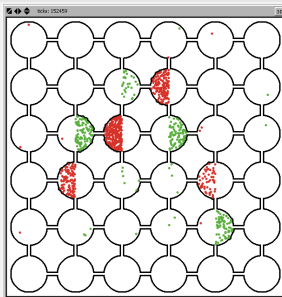


Current Work (3)

Further Steps

- Extension of the base algorithm explicitly for general-topology networks via the adoption of:
 - an aging mechanism
 - a rejection mechanism

Early Results



SwarmLinda: Introduction (1)

Scenario

- Coordination of active entities in open distributed scenarios by tuple-space systems
- Specific issue: tuple distribution and organization in a network of distributed tuple spaces

Why is tuple distribution and organization an important issue?

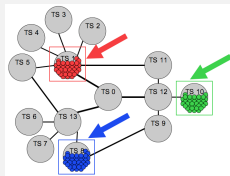
- It affects the scalability of a system ...
- ... indeed, if tuples are organized in a specific way, we can obtain a better system's scalability
- For instance, if similar tuples are kept close to each other it is easier for processes to find the specific tuple they need!



SwarmLinda: Introduction (2)

Tuple Organization: important issues

- Keeping similar tuples close to each other makes processes too dependent on aggregating nodes
- This leads to **over-clustering**: system's robustness is affected



Objectives

- Exploiting **self-organization** to build a tuple-space architecture with a scalable tuple-distribution mechanism
- Showing that this mechanism is self-organizing



A Strategy to Organize Tuples

Developed by taking inspiration from ant's brood sorting

Aim of the Strategy

- Storing a tuple in a node with a high concentration of similar tuples
- Metaphor:
 - **outs** are **ants** carrying items and searching for a suitable node where to drop food
 - **tuples** are **items** of different kind carried by ants



How Tuple Distribution Works (0)

Phases

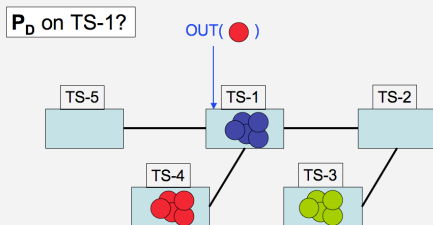
- Our tuple-distribution strategy is composed of **two phases**:
 - **Storing decision phase**: does a tuple have to be stored in the current node?
 - **Movement phase**: what is a good neighbor where to move a tuple?

Similarity Function

- How **similar** are two tuples tu and te ?
 - $\delta(tu, te) = [0, 1]$



How Tuple Distribution Works (1)

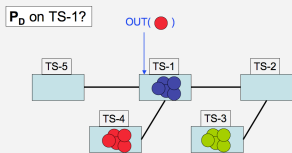


Storing Decision Phase (1)

- **Drop probability** $P_D = \left(\frac{F}{F+K} \right)^2$:
 - Concentration of tuples similar to tu : $F = \sum_{\forall t \in TS} \delta(tu, t)$
 - K is the number of hops left to an *out*-ant before storing the carried tuple tu
 - K is initially set to $Step_{MAX}$



How Tuple Distribution Works (1)



- Drop probability $P_D = \left(\frac{F}{F+K}\right)^2$:
 - Concentration of tuples similar to tu : $F = \sum_{\forall t \in TS} \delta(tu, t)$
 - K is the number of hops left to an *out-ant* before storing the carried tuple tu
 - K is initially set to $Step_{MAX}$

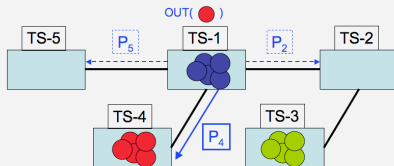
Storing Decision Phase (2)

- K implements an **aging mechanism**
 - the aging mechanism avoids to have *out-ants* wandering forever in a network without being able to store the carried tuple.
- $Step_{MAX}$ is the maximum number of tuple spaces an *out-ant* can visit
 - each time an *out-ant* is not able to store the carried tuple in the current tuple space, K is decreased by 1 and the *out-ant* moves to a neighboring tuple space according to a **movement phase**



How Tuple Distribution Works (2)

Which neighbor to move to ?



Movement Phase

- \forall neighbor j , the **movement probability** P_j is determined:

$$P_j = \frac{F_j}{\sum_{i=1}^n F_i}$$

- F_j : concentration of tuples similar to tu in tuple space j
- $\sum_{i=1}^n F_i$: concentration of tuples similar to tu in the neighborhood of the current tuple space



Simulations

Network Topology

- Tested the strategy on networks featuring a scale-free topology
 - this topology often recurs in real computer networks
- Test instances generated by *Barabási and Albert Scale-Free Model Algorithm*

Test Instances

- Two scale-free test instances composed of
 - 30 and 100 tuple spaces
- Tuples belonging to 4 different templates
- Insertion of 60 tuples per tuple space (15 per tuple template)
- $$\delta(tu, te) = \begin{cases} 1 & \text{template}(tu) = \text{template}(te) \\ 0 & \text{otherwise} \end{cases}$$

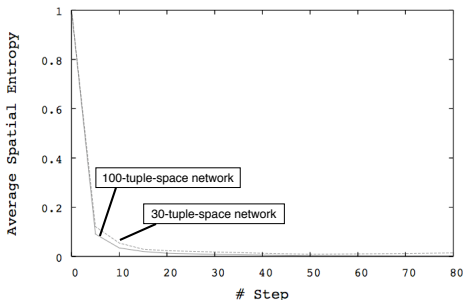


Simulation Methodology

- For each test instance:
 - performed simulations for different values of $Step_{MAX}$ in the range $[0,80]$
- As a measure of the order degree of the information stored in a network we adopted the concept of **spatial entropy H**
- H can be viewed as a measure of **information chaos**
- H features values bounded between 0 and 1:
 - 0 means *complete order* or even better *complete information clustering*
 - 1 means *complete chaos*
- For each value of $Step_{MAX}$:
 - run a series of 20 simulations
 - calculated the value of **average spatial entropy H_{avg}** resulting as average of the single H values obtained by each simulation



Simulation Results



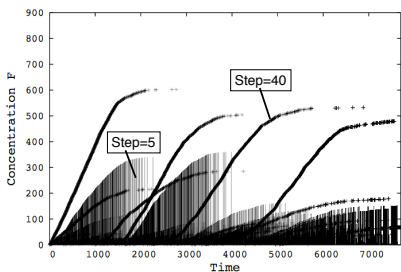
Observations

- Greater values of $Step_{MAX}$ lead to lower values of H_{avg} !
 - *out*-ants can have better chances to explore the entire network
- For $Step_{MAX} > 20$ we have a *horizontal asymptote* = 0.014
 - (quasi-)complete clustering!



Is our Approach Self-Organizing? (1)

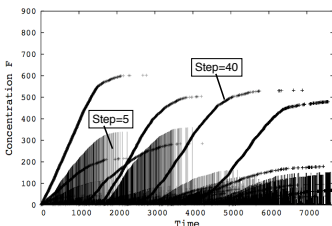
- How to know that our strategy is truly self-organizing?
- A first clue:



- This is a simulation trace showing the trend of concentration F ...
- ... F here refers to the concentration of tuples similar to the current one in the tuple space where the tuple is eventually stored
- The chart shows the trend of F for two different values of $Step_{MAX}$



Is our Approach Self-Organizing? (2)



Meaning

- We know neither the tuple space where a tuple is stored nor the template of that tuple, nonetheless . . .
- . . . it is easy to see the formation of clusters per tuple template
- Clustering **emerges** from an initially-empty network
- Such behavior is the result of **local interactions** among each tuple space and its neighbors



Proving the Self-Organizing Nature of our Approach (1)

- **Camazine** argumentation (from *Self-Organization in Biological Systems*):
 - Many cases of collective behavior (e.g. wood lice and stable fly) feature organism clustering which arises due to the presence of an **external cue** acting as **environmental template** for the aggregation process
 - In such a situation, if we remove the environmental template, clustering does not occur anymore



Proving the Self-Organizing Nature of our Approach (2)

Camazine's Method to Understand Self-Organization

1. Finding an environmental template suspected to lead the aggregation process
2. Making experiments with initial conditions featuring the presence of such environmental templates: a false self-organizing process shows **insensitivity to different initial conditions**
3. Making experiments with no environmental templates: if clustering occurs, then we have a good indication of the self-organizing nature of our approach

Camazine's Method Applied to our Case

1. Finding an environmental template: **presence of already-formed clusters in the network**
2. Making experiments with initial conditions featuring the presence of such environmental templates: **experiments made by using already-formed clusters of different sizes**
3. Making experiments with no environmental templates; if clustering occurs, then we have a good indication of the self-organizing nature of our approach: **made experiments with initially-empty networks**

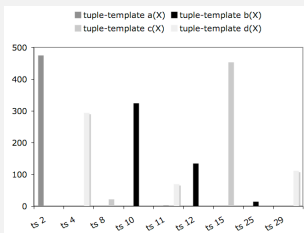


Simulation Results (1)

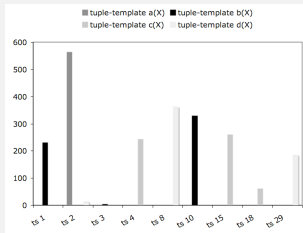
- Simulations executed on a 100-tuple-space network with $Step_{MAX} = 40$ and tuples of 4 different templates

Already-Formed Clusters: One Cluster per Tuple Template

- cluster size: 10



- cluster size: 100



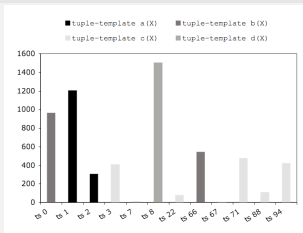
- Clustering occurs and also shows a **dependence** on different cluster sizes (different initial conditions)



Simulation Results (2)

- Simulations executed on a 100-tuple-space network with $Step_{MAX} = 40$ and tuples of 4 different templates

Network Initially Empty



- H corresponding to this simulation: 0.0054

Clustering still occurs even without the presence of clusters!



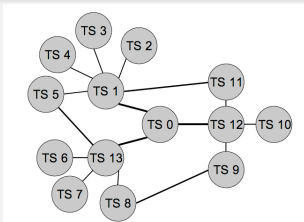
Simulation Results: Review

- Clustering occurs both with and without the presence of clusters already-formed in the network
- With clusters:
 - clustering evolution is driven not only by interactions among *out*-ants, but also by clusters
 - it is likely to get aggregation in those tuple spaces featuring already-formed clusters
- With no clusters:
 - clustering still occurs only as a result of interactions among *out*-ants



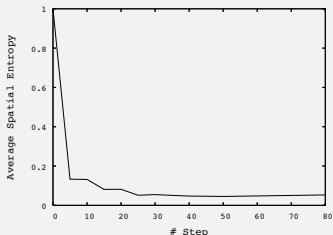
Over-Clustering (1)

Test Instance



- 14 tuple spaces
- Tuples belonging to 4 different templates
- Insertion of 200 tuples per template
- $\delta(tu, te) = \begin{cases} 1 & \text{template}(tu) = \text{template}(te) \\ 0 & \text{otherwise} \end{cases}$

Results: Spatial Entropy

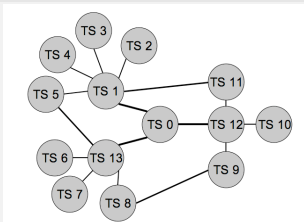


- 20 simulations per $Step_{MAX}$ value

Almost complete clustering!

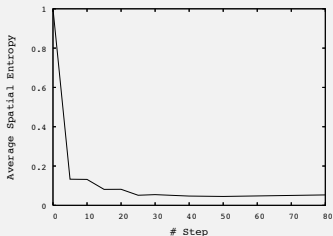
Over-Clustering (2)

Test Instance



- 14 tuple spaces
- Tuples belonging to 4 different templates
- Insertion of 200 tuples per template
- $\delta(tu, te) = \begin{cases} 1 & \text{template}(tu) = \text{template}(te) \\ 0 & \text{otherwise} \end{cases}$

Results: Spatial Entropy



OVER-CLUSTERING!

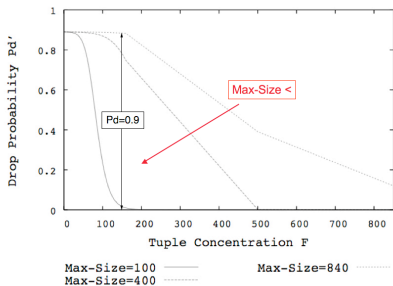


A Solution to Over-Clustering

Max-Size

- For each node and tuple template, introduction of a *Max-Size* parameter in order to:
 - Regulate the maximum size of the clusters in the network by
 - ...
 - ... modifying P_D when a cluster gets as big as *Max-Size*

How Max-Size Works



- A **sigmoid function** affects P_D
- The smaller *Max-Size* the higher the slope
 - Fixing a value of concentration F , the smaller *Max-Size* the smaller P_D

ANTI OVER-CLUSTERING!



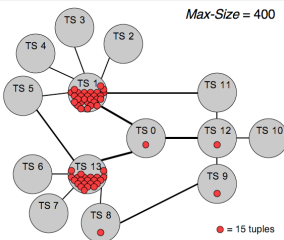
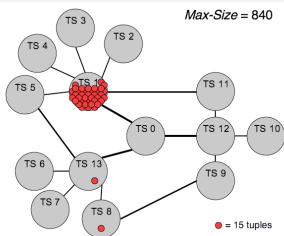
Anti-Over-Clustering Simulation

- Same test instance
- Insertion of tuples belonging to one template
 - Occurrence of 60 *out* operations per tuple space: 840 tuples, in total



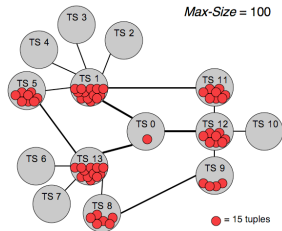
Anti-Over-Clustering Simulation: Results

Tuple Organization in the Network



← From quasi-complete clustering ...

... to clusters of smaller sizes for smaller values of *Max-Size* ↓



Conclusion and Future Work

Conclusion

- Developed a self-organizing strategy to solve tuple-distribution problem
 - made experiments with larger scale-free networks (> 1000 nodes)
- Provided an argumentation to show the self-organizing nature of the strategy

Future Work

- Making experiments on other network topologies
- Solving the over-clustering problem: for instance by experimenting with dynamic similarity functions
- Devising a tuple-retrieval strategy based on what we have gotten regarding tuple-distribution



Publications I



Casadei, M., Gardelli, L., and Viroli, M. (2006a).

A case of self-organising environment for mas: the collective sort problem.

In Omicini, A., Dunin-Kępicz, B., and Padget, J. A., editors, *4th European Workshop on Multi-Agent Systems (EUMAS 2006)*, number 223 in CEUR Workshop Proceedings, Lisbon, Portugal. Sun SITE Central Europe, RWTH Aachen University.



Casadei, M., Gardelli, L., and Viroli, M. (2006b).

Collective sorting tuple spaces.

In Paoli, F. D., Stefano, A. D., Omicini, A., and Santoro, C., editors, *WOA*, volume 204 of *CEUR Workshop Proceedings*. CEUR-WS.org.



Publications II



Casadei, M., Gardelli, L., and Viroli, M. (2006c).
Simulating emergent properties of coordination in Maude: the
collective sorting case.

In Canal, C. and Viroli, M., editors, *5th International
Workshop on Foundations of Coordination Languages and
Software Architectures (FOCLASA'06)*, pages 57–75,
CONCUR 2006, Bonn, Germany. University of Málaga, Spain.
Proceedings.



Casadei, M., Gardelli, L., and Viroli, M. (2007a).
Simulating emergent properties of coordination in maude: the
collective sort case.

Electr. Notes Theor. Comput. Sci., 175(2):59–80.



Publications III



Casadei, M., Menezes, R., Tolksdorf, R., and Viroli, M.
(2007b).

On the problem of over-clustering in tuple-based coordination systems.

In *SASO*, pages 303–306. IEEE Computer Society.



Casadei, M., Menezes, R., Viroli, M., and Tolksdorf, R.
(2007c).

Self-organized over-clustering avoidance in tuple-space systems.

In *the 2007 IEEE Congress on Evolutionary Computation*, Singapore.



Publications IV



Casadei, M., Menezes, R., Viroli, M., and Tolksdorf, R. (2007d).

A self-organizing approach to tuple distribution in large-scale tuple-space systems.

In Hutchison, D. and Katz, R. H., editors, *IWSOS*, volume 4725 of *Lecture Notes in Computer Science*, pages 146–160. Springer.






Casadei, M., Menezes, R., Viroli, M., and Tolksdorf, R. (2007e).

Using ant's brood sorting to increase fault tolerance in linda's tuple distribution mechanism.

In Klusch, M., Hindriks, K. V., Papazoglou, M. P., and Sterling, L., editors, *CIA*, volume 4676 of *Lecture Notes in Computer Science*, pages 255–269. Springer.



Publications V

-  Casadei, M., Omicini, A., and Viroli, M. (2007f).
Prototyping A&A ReSpecT in Maude.
In Canal, C., Poizat, P., and Viroli, M., editors, *6th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'07)*, pages 133–148, CONCUR 2007, Lisbon, Portugal. Proceedings.
-  Casadei, M., Omicini, A., and Viroli, M. (2008).
Prototyping a&a ReSpecT in Maude.
In *To be published in Electr. Notes Theor. Comput. Sci.*
-  Gadelli, L., Viroli, M., Casadei, M., and Omicini, A. (2007).
Designing self-organising environments with agents and artifacts: A simulation-driven approach.
International Journal of Agent-Oriented Software Engineering (IJAOSE), 2(2).



Publications VI



Gardelli, L., Viroli, M., and Casadei, M. (2006a).

Engineering the environment of self-organising multi-agent systems exploiting formal analysis tools.

In *Congresso AICA 2006*, Cesena - Italy. AICA - Associazione Italiana per l'Informatica e il Calcolo Automatico.






Gardelli, L., Viroli, M., and Casadei, M. (2006b).

On engineering self-organizing environments: Stochastic methods for dynamic resource allocation.

In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2006)*, pages 96–101, AAMAS 2006, Hakodate, Japan.



Publications VII

-  Gardelli, L., Viroli, M., Casadei, M., and Omicini, A. (2006c).
Designing self-organising mas environments: The collective sort case.
In *Electr. Notes Theor. Comput. Sci.*, pages 254–271.
-  Viroli, M., Casadei, M., and Gadelli, L. (2007a).
On the collective sort problem for distributed tuple spaces.
Paper Invited to Science of Computer Programming **currently under review**.
-  Viroli, M., Casadei, M., and Gardelli, L. (2007b).
A self-organising solution to the collective sort problem in distributed tuple spaces.
In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 354–359, New York, NY, USA. ACM Press.

