Designing Self-Organising MAS Environments: the Collective Sort Case

Luca Gardelli, Mirko Viroli, Matteo Casadei, and Andrea Omicini

Alma Mater Studiorum—Università di Bologna
via Venezia 52, 47023 Cesena, Italy
{luca.gardelli, mirko.viroli, m.casadei, andrea.omicini}@unibo.it

Abstract. Self-organisation is being recognised as an effective conceptual framework to deal with the complexity inherent to modern artificial systems. In this article, we explore the applicability of self-organisation principles to the development of multi-agent system (MAS) environments. First, we discuss a methodological approach for the engineering of complex systems, which features emergent properties: this is based on formal modelling and stochastic simulation, used to analyse global system dynamics and tune system parameters at the early stages of design. Then, as a suitable target for this approach, we describe an architecture for self-organising environments featuring artifacts and environmental agents as fundamental entities.
As an example, we analyse a MAS distributed environment made of tuple spaces, where environmental agents are assigned the task of moving tuples across tuples spaces in background and according to local criteria, making complete clustering an emergent property achieved through self-organisation.

1 Introduction

The typical MAS scenario involves a set of autonomous situated entities interacting with each other and exploiting resources in the environment to achieve a common goal [1, 2]. When designing the MAS environment, other than functional properties concerning the available services offered to agents, one has also to consider non-functional properties. First, it is crucial to balance the computational power devoted to the provision of such services: several maintenance activities must be performed on-line and in background to guarantee a certain level of quality of service at low computational cost. Moreover, the design of environment should feature the emergence of quality properties: since the dynamics of interactions with agents cannot be fully predicted, several unexpected situations have to be handled automatically and effectively.

As standard optimisation techniques (e.g. in resource allocation) are offline and possibly computationally expensive, we need to rely on different approaches. Self-organisation theory is a rich source of inspiration for sub-optimal strategies that can be performed on-line, like every natural system does, and that require little computational power, e.g. because of local communications. For these reasons, we look for a methodological approach for the development of MAS environments that can systematically adopt self-organisation principles, as a means to more effectively and efficiently provide agents with useful services.

We consider the A&A (agents & artifacts) meta-model as our reference model for MAS [3, 4]. *Artifacts* are environment abstractions encapsulating resources and services provided by the environment to agents: agents exploit artifacts in order to achieve individual as well as social goals. Following some early exploration of self-organising systems [5, 6], we see the environment as a set of artifacts, exploited by *user agents* and managed by *environmental agents*, the latter having the responsibility of regulating artifact behaviour and state, and making interesting quality properties emerge whenever needed. This vision is of course shared by the Autonomic Computing initiative [7]: in particular Autonomic Computing emphasises the need for artificial systems to self-configure, self-protect, automatically recovers from errors (self-healing) and self-optimise.

In order to focus on the the application of self-organisation principles to the engineering of MAS environment, we devise a methodological approach based on formal modelling and stochastic simulations. In spite of the difficulty of designing "emergence", this approach allows us to preview global system dynamics and tune the system model until possibly converging to a system design matching the expected requirements.

As a case study we discuss an application called *collective sort*, which is a generalisation of the brood sorting problem of swarm intelligence [8] to a distributed tuple space scenario [9]. User agents interact with each other by putting and retrieving information on the environment in the form of tuples: the distributed environment consists of a set of artifacts resembling tuple spaces, which are kept ordered by environmental agents exploiting self-organising techniques. The objective is to devise a fully-distributed, swarm-like strategy for clustering tuples according to their type (as a tuple template), so that tuples with same type are aggregated into the same, unique tuple space. In this case, it is the agent-environment coupling that creates the feedback loop supporting emergence of ordering, by balancing between positive and negative feedback [10]. We provide an architecture and simulation results for this problem, showing the effectiveness of the strategy proposed.

This paper is structured as follows: in Section 2 we discuss our methodological approach for engineering self-organising systems, and its application to our basic architecture for MAS environments. We then apply the methodology—which is based on the three steps of *modelling*, *simulation*, and *tuning*—to the collective sort scenario. Specifically, in Section 3 we devise a model for a self-organising environment, in Section 4 we show simulation results from the system model in order to investigate global dynamics, and then in Section 5 we tune the strategy for achieving better performance, relying on load-balancing techniques. In Section 6 we discuss some related work, and finally in Section 7 we conclude by pointing at some possible future developments.

2 A Design Approach for Self-Organising MAS Environments

In the MAS community, several engineering methodologies have been developed [11–13], but they typically do not face some of the most important issues in self-organisation: Given a problem, (how) can we design the individual agent's behaviour in order to let the desired property emerge? Once a candidate behaviour is specified, (how) can we guarantee that the specific emergent property will appear?

We recognise in general two approaches to tackle the above issues: (i) analysing the problem and designing by decomposition an *ad-hoc* strategy that will solve it; (ii) observing a system that solves a similar problem, and trying to reverse-engineer the strategy. The former approach is applicable only to a limited set of simple scenarios, since the non-linearity in the rules makes the results quite difficult to predict—indeed, it hardly scales with the complexity of a problem [14]. A self-organisation viewpoint suggests that the latter approach might be more fruitful: indeed, several tasks accomplished by social systems have their counterpart in computer science algorithms and architectures [8, 15]. Unfortunately, strategies for specific problems are often unknown, so they are inferred by modifying the original models: but then, how can we guarantee that such modifications would not produce side-effects, namely, behaviours that significantly differ from the expected one?

In general, ensuring that a design leads to the desired dynamics is still an open issue. Although it is possible to verify properties of a deterministic model via automated tools and techniques—e.g. with model checking—, as soon as stochastic aspects enter the picture, verification becomes more difficult—existing works in this context appear to be somehow immature. Hence, although applying model checking techniques is part of our medium-term research objectives—see a discussion in Section 7—it is necessary to resort to a different methodology for analysing the behaviour and qualities of a design.

Then, in the remainder of this section, we describe our methodological approach and, next, how concepts from the methodology map onto design abstractions.

2.1 A Methodological Approach based on Formal Modelling and Stochastic Simulations

Our approach starts from the deliverable of analysis, and has the goal of devising out an *early design* of the system, to be later detailed and implemented. However, we do not aim at devising a complete methodology: instead of covering the complete development process and tackling all functional and non-functional aspects, we rather intend to use it in combination with other existing AOSE methodologies. So, working at the early design phase, in our approach we aim at evaluating several strategies and their crucial parameters, with the goal of discovering the one that could provide the quality attributes required for the application at hand.

In particular, our approach is articulated around three activities:

Modelling — to develop an abstract formal specification of the system;
Simulation — to qualitatively investigating the dynamics of the system;
Tuning — to change model parameters to adjust system behaviour.

In the *modelling* activity, we formulate a few strategies that seemingly fit the behaviour to be implemented: as far as self-organisation is concerned, natural models are an obvious source of inspiration. Such strategies should form an abstract model of the system for possible architectural solutions. Specifically, in the case of self-organising systems where complex patterns arise from low-level interaction, selective models can make us focus on the properties of interest. To enable further automatic elaborations—such as simulation and automatic verification of properties—these descriptions should be provided in a formal language, which promotes unambiguity and precise selection of the features to be modelled and those to be abstracted away. Although we do not endorse any particular language, given the nature of the target systems we prefer languages that easily and concisely express notions such as distribution, concurrency, communication and compositionality [5]. Moreover, they must be able to deal with stochastic aspects, which are required in order to properly abstract over unpredictability of certain behaviours and events.

The deliverable of the modelling phase is then a formal specification. In the subsequent activity, this specification is used in combination with simulation tools in order to generate *simulation* traces. Simulation is a very useful tool able to provide a first feedback about the suitability of a solution: for this purpose, tools like SPiM [16] for the stochastic π-calculus have been proved to be quite useful [5, 6]. Tools that do not directly support simulation features could be extended by using e.g. the Gillespie algorithm [17], as done for the SPiM simulator [18]. Some general-purpose engine can also be developed to this end, like e.g. the MAUDE term rewriting system module described in [9].

Since self-organising systems tend to display different qualitative dynamics depending on initial conditions, it may happen that simulations of the current design do not exhibit interesting behaviours: the model is then to be tuned until the desired qualitative dynamics is reached. Thus, the parameters employed works as a coarse set of parameters for implementation, while fine tuning is delayed until the actual implementation has been developed. The tuning process may end up with unrealistic values for parameters, or simply may not converge to the required behaviour, meaning that the chosen model cannot be implemented in a real scenario: in this case another approach should be tested, going back to the modelling activity.

2.2 A Basic Architecture for Self-Organising Environments

The multi-agent paradigm is a natural choice for modelling natural systems and developing nature-inspired, artificial ones. In particular, in this paper we adopt the agents & artifacts meta-model (A&A) for MAS, where MAS are modelled

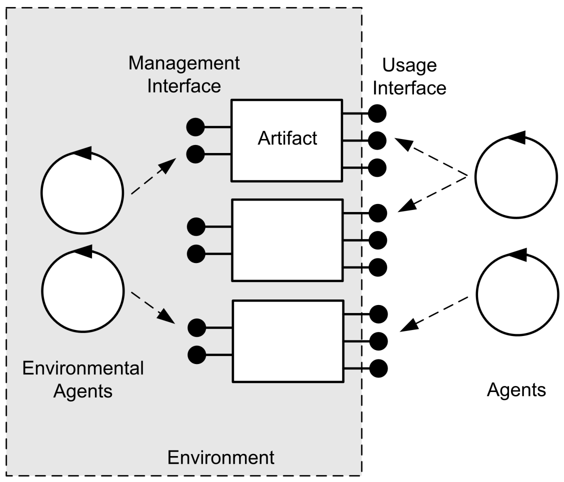

Fig. 1. Basic architecture for a MAS featuring environmental agents as artifacts administrators.

and engineered based on two fundamental abstractions: *agents* and *artifacts* [3, 19]. Agents are the (pro-)active entities encapsulating control: they are in charge of the goals/tasks that altogether build up the whole MAS behaviour. Artifacts are instead the passive, reactive entities in charge of the services and functions that make individual agents work together in a MAS, and which shape agent environment according to the MAS needs. Other than being an interpretation means for a number of existing concepts in MAS—coordination media, e-institutions, stigmergic fields, web services [3]—this meta-model has also an impact on practise: it is e.g. the basis of the CArtAgO project for developing a general-purpose infrastructure for MAS environments [4].

Based on this meta-model, we focus on the development of environments featuring self-organisation properties, accordingly propose an architectural solution, and then discuss the impact on methodological issues. From the viewpoint of *(user) agents*, that exploit the services provided by artifacts, we see the environment as composed by a set of *environmental agents*, other than the artifacts themselves. As depicted in Figure 1, artifacts exhibit a usage interface which is accessible to user agents, and which provides the artifact services, while the management interface is accessible only to environmental agents, and provides features related to controllability and malleability of artifacts. Environmental agents are in charge of managing resources—say, in an Autonomic Computing style—by adjusting artifact behaviour and status, and by performing periodic administration tasks, possibly taking part in the self-organisation process that the environment should globally exhibit. In particular, this architecture supports the positive/negative feedback loop together with agents: since self-organisation is an active process, it is often the case that artifacts alone cannot close the feedback loop because of the lack of pro-activity, which is instead featured by environmental agents.[1]

2.3 Relating the Methodology to the Architecture

Modelling When modelling an environment according to our architecture, we have to consider three elements: *(i)* the user agents requesting services, *(ii)* the artifacts providing the interface to the services, and *(iii)* the environmental agents administering the artifacts and driving the self-organisation process (when needed).

Starting from user agents, we observe that they cannot be modelled as fully predictable entities because of their autonomy, and because their behaviour is mostly unknown to the environment designer: abstraction is a necessary process here to cope with the peculiarities of agents to come. Hence, stochastic models are to be used in order to abstract away from agent internal behaviour, simply exposing timing and probability aspects—such as e.g. at which rate they interact with artifacts.

On the other hand, the behaviour of artifacts is predictable by definition, for artifacts automatise the service: hence, in the modelling stage it is typically possible to precisely model their internal state and describe their step-by-step behaviour.

Finally, environmental agents lie somehow in the middle: because they are strongly coupled with artifacts, they are typically designed along with them. Hence, in spite of their autonomy, we can make quite reliable predictions about the behaviour of environmental agents, though stochastic aspects can be anyway useful to model their effect on artifacts.

If the system is designed to exhibit emergent properties, then user and environmental agents are necessarily functionally coupled: such a coupling is required for the positive/negative feedback loop, in that the result of actions performed by user agents eventually triggers a response by environmental agents. Hence, some assumptions about the nature of feedback have to be made: however, these assumptions are not too restrictive, since the set of services offered by the environment is limited.

Simulating When it comes to simulating, it is necessary to provide a set of operating parameters for the system modelled. It is worth noting that, for the simulation to be meaningful, parameters should represent actual values, otherwise it is not possible to decide about the feasibility of the solution: hence,

[1] It is worth noting that our view of MAS environment should not be considered as a departure from the original idea of the A&A model, where the environment is made by artifacts alone. Indeed, drawing the boundary of the environment is a subjective task: as far as we call "environment" what is outside a particular subset of agents, it could be naturally seen as made of artifacts and by the remaining agents. So, our architecture here is to be considered as a possible specialisation of the A&A meta-model to the case of self-organising environments.

devising such parameters is probably the most crucial aspect in the simulation stage. It is then possible to investigate the system global dynamics before actually implement it. In order to preview certain behaviours, it is necessary to sweep through unbounded parameters: tweaking parameters is almost unavoidable since—working with complex systems—small modifications likely lead to qualitatively very different results.

While it is possible to precisely characterise the behaviour of artifacts and environmental agents, we cannot foretell user agents behaviour: then, stochastic and probabilistic models can abstract from those details, though introducing simplifying assumptions that should be later validated. What we actually need is to devise likely scenarios—completeness is typically unfeasible—each of which characterised by a statistical description of user agents' interaction with artifacts. For example, if system behaviour depends on the distribution in time of service requests, it is necessary to investigate several scenario, e.g. even, random, burst distribution and the like. The goal of each simulation is then to understand whether the behaviour of environmental agents is suitable to achieve the properties required.

Tuning While parameters for artifacts and user agents likely reflect an actual implementation, environmental agents are the real place where tuning occurs: working parameters of environmental agents are to be adjusted, keeping them within the range of physical feasibility, until the system exhibits the desired behaviour. If the required behaviour is not reached with a valid set of parameters, it is then necessary to *tune the model* until such a behaviour is displayed. Modifications usually involve aspects linked to locality and computational cost, that is, in order to make the system converge to the desired state it might be necessary to extended the locality boundaries to let an environmental agent gather further information, or to adopt more sophisticated strategies. However, it is worth noting that pursuing that line too forward moves us away from what is the *self-organisation philosophy* towards the more traditional approach of devising optimal solutions—which typically require too much computational price, and are therefore unsuitable e.g. for on-line tasks.

In the end, it may happen that, despite these modifications, the desired behaviour is not met: then, we have to look for different models and strategies, going back to the modelling stage.

3 Step 1: Modelling the Collective Sort Strategy

In the rest of the article, we apply our methodology to a case we name *collective sort* [9]:

1. we start by defining the problem and identifying key aspects of a possible solution;
2. the strategy is translated into a suitable formalism, enabling the execution of stochastic simulations and analysis of results;

3. variations on the strategy can be evaluated by tuning system model so as to increase the expected performance.

3.1 Motivation and Problem Description

We aim at developing an environment that keeps similar tuples—i.e. having the same template—clustered in the same tuple space, uniformly distributing the load across the nodes where tuple spaces reside. In several scenarios, sorting tuples may increase the overall system performance. For instance, it can make it easier for an agent to find an information of interest based on its previous experience: the probability of finding an information where a related one was previously found is high. Moreover, when tuple spaces contain tuples of one kind only, it is possible to apply aggregation techniques to improve performance, and it is generally easier to manage and achieve load-balancing.

Increasing system order however comes at a computational price. Since we want the sorting process to take place on-line and in background, we look for suboptimal algorithms, which are able to guarantee a certain degree of ordering in time without requiring too much computational power. To this purpose, we look at existing self-organising systems to find one that exhibits this capability of sorting items: in particular, we identify a suitable solution in the *brood sorting* problem of swarm intelligence [8]. In brood sorting, ants cluster broods that were initially dispersed in the environment: although the actual behaviour of ants is still not fully understood, there are several models that are able to mimic the dynamics of the system. Ants behaviour is modelled by two probabilities [8], respectively, the probability to pick up P_p and drop P_d an item

$$P_p = \left(\frac{k_1}{k_1 + f}\right)^2, \quad P_d = \left(\frac{f}{k_2 + f}\right)^2, \tag{1}$$

where k_1 and k_2 are constant parameters, and f is the number of items perceived by an ant in its neighbourhood and may be evaluated with respect to the recently encountered items. To evaluate the system dynamics, it could be useful to provide a metric for system order: such an estimation can be obtained by measuring the spatial entropy, as done e.g. in [20]. Basically, the environment is subdivided into nodes, and P_i is the fraction of items within a node, hence the local entropy is $H_i = -P_i \log P_i$. The sum of H_i having $P_i > 0$ gives an estimation of the order of the entire system, which is supposed to decrease in time, hopefully reaching zero (complete clustering).

We aim at generalising this approach for an arbitrary number of item kinds, and we call it *collective sort*. We conceive this environment itself as a MAS, i.e. made of artifacts and environmental agents: the goal of these agents is to collect and move tuples across the environment so as to order them according to an arbitrary shared criterion. We consider the case of a fixed number of tuple spaces hosting tuples of a known set of tuple types: the goal of agents is to move tuples from one tuple space to another until the tuples are clustered within different tuple spaces according to their tuple type.

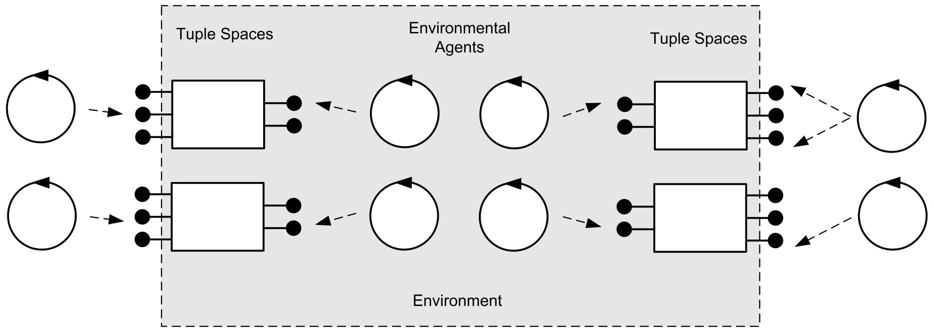

Fig. 2. The basic architecture consists in a set of environmental agents moving tuples across tuple spaces.

3.2 A Solution to the Collective Sort Problem

The basic bricks are user agents and tuple spaces (realised through artifacts): user agents are allowed to read, insert and remove tuples in the tuple spaces. Transparently to user agents, the environment provides a sorting service in order to maintain a certain degree of ordering of tuples in tuple spaces. This functionality is realised by a class of environmental agents that is responsible for the sorting task. Hence, each tuple space is associated with one or more environmental agents—see Figure 2—whose task is to compare the content of the local tuple space against the content of another tuple space in the environment, and possibly move some tuple. Since we want to perform this task on-line and in background we cannot compute the probabilities in Equation 1 to decide whether to move or not a tuple: the approach would not be scalable since it requires to count all the tuples for each tuple space, which might not be practical.

Hence, we devise a strategy based on tuple sampling, and suppose that tuple spaces provide for a reading primitive we call `urd`, *uniform read*. This is a variant of the standard `rd` primitive that takes a tuple template and yields any tuple matching the template: primitive `urd` instead chooses the tuple in a probabilistic way among all the tuples that could be returned. For instance, if a tuple space has 10 copies of tuple $t(1)$ and 20 copies of tuple $t(2)$ then the probability that operation $urd(t(X))$ returns $t(2)$ is twice as much as $t(1)$'s. As standard Linda-like tuple spaces typically do not implement this variant, it can e.g. be supported by some more expressive model like ReSpecT tuple centres [21]. When deciding to move a tuple, an agent working on the tuple space TS_S follows this agenda:

1. it draws a destination tuple space TS_D different from the source one TS_S;
2. it draws a kind k of tuple;
3. it (uniformly) reads a tuple T_1 from TS_S;
4. it (uniformly) reads a tuple T_2 from TS_D;
5. if the kind of T_2 is k and it differs from the kind of T_1, then it moves a tuple of the kind k from TS_S to TS_D.

The point of last task is that if those conditions hold, then the number of tuples k in TS_D is more likely to be higher than in TS_S, therefore a tuple could/should be moved. It is important that all choices are performed according to a uniform probability distribution: while in the steps 1 and 2 this guarantees fairness, in steps 3 and 4 it guarantees that the obtained ordering is appropriate.

It is worth noting that the success of this distributed algorithm is an emergent property, affected by both probability and timing aspects. Will complete ordering be reached starting from a completely chaotic situation? Will complete ordering be reached starting from the case where all tuples occur in just one tuple space? And if ordering is reached, how many moving attempts are globally necessary? These are the sort of questions that could be addressed at the early stages of design, thanks to a simulation tool.

4 Step 2: Simulating the Collective Sort Strategy

In this section we briefly describe simulation results obtained by using the MAUDE tool. MAUDE is a high-performance reflective language supporting both equational and rewriting logic specifications, for specifying a wide range of applications [22]. Since MAUDE does not directly provide any facility for simulation purposes, we developed a general simulation framework for stochastic systems: the idea of this tool is to model a stochastic system by a labelled transition system where transitions are of the kind $S \xrightarrow{r:a} S'$, meaning that the system in state S can move to state S' by action a, where r is the *(global) rate* of action a in state S—that is, its occurring frequency. We do not describe here the simulation framework since that would requires a separate treatment: interested readers can refer to [9] for more details about MAUDE and a comprehensive description of our framework.

Given the strategy described in Section 3.2, we translated it into the MAUDE syntax. Our reference case sticks to the case where four tuple spaces exist, and four tuple kinds are subject to ordering: we represent the distributed state of a system in MAUDE using a syntax of the kind:

```
< 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]) >
```

It expresses the fact that we work with the tuple kinds `'a`, `'b`, `'c`, and `'d`, and with the tuple spaces identifiers `0`, `1`, `2`, and `3`. The content of a tuple space `0` is expressed as `< 0 @ ('a[100])|('b[100])|('c[10])|('d[10]) >`, meaning that we have 100 tuples of kind `'a`, 100 of kind `'b`, 10 of `'c`, and 10 of `'d`. The formal definition of agents agenda is defined in terms of simple transition rules. The chart in Figure 3 reports the dynamics of the winning tuple in each tuple space, showing e.g. that complete sorting is reached at different times in each space. The chart in Figure 4 displays instead the evolution of the tuple space `0`: notice that only the tuple kind `'a` aggregates here despite its initial concentration was the same of tuple kind `'b`.

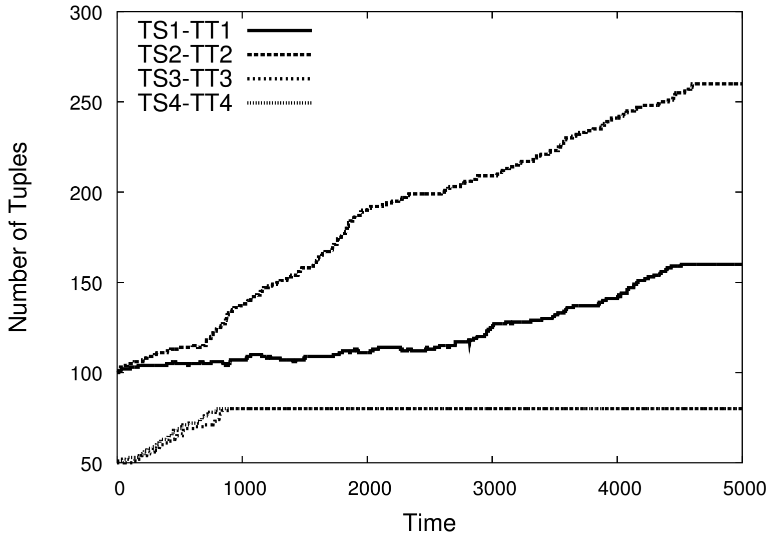

Fig. 3. Dynamics of the winning tuple in each tuple space: notice that each tuple aggregates in a different tuples space.

Although it would be possible to make some prediction, we do not know in general which tuple space will host a specific tuple kind at the end of sorting: this is an emergent property of the system and is the very result of the *interaction* of environmental agents through the tuple spaces. It is interesting to analyse the trend of the entropy of each tuple space as a way to estimate the degree of order in the system through a single value: since the strategy we simulate is trying to increase the inner order of the system we expect the entropy to decrease, as it actually happens as shown in Figure 5.

5 Step 3: Tuning the Collective Sort Strategy

Such a strategy based on constant rates is not very efficient, since agents are assigned to a certain tuple space and keep working also if the tuple space is already ordered. We may exploit this otherwise wasted computational power by assigning idle agents to unordered tuple spaces, or rather, by dynamically adapting the working rates of agents. This alternative therefore looks suited to realise a strategy to more quickly reach the complete ordering of tuple spaces.

In order to adapt the agents rate, we adopted spatial entropy as a measure of system order. If we denote with q_{ij} the amount of tuples of the kind i within the tuple space j, n_j the total number of tuples within the tuple space j, and

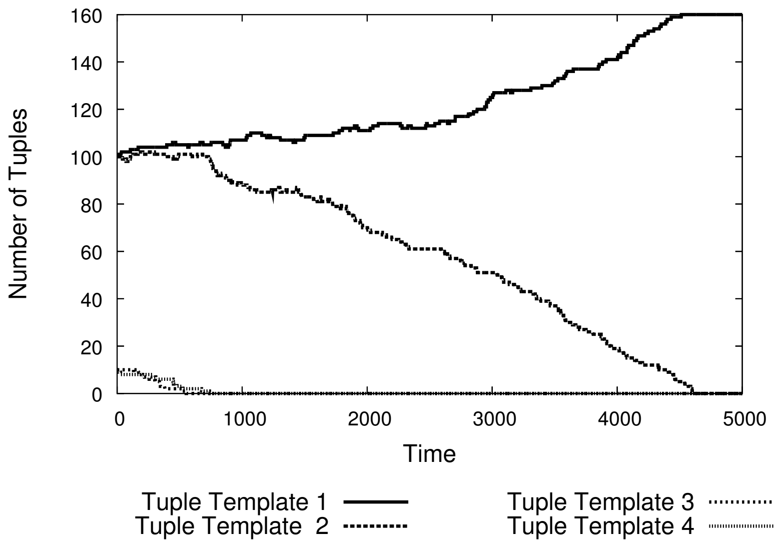

Fig. 4. Dynamic of tuple space 0: notice that only one kind of tuple aggregates here.

k the number of tuple kinds, then, the entropy associated with the tuple kind i within the tuple space j is

$$H_{ij} = \frac{q_{ij}}{n_j} \log_2 \frac{n_j}{q_{ij}} \tag{2}$$

and it is easy to notice that $0 \leq H_{ij} \leq \frac{1}{k} \log_2 k$. We want to express now the entropy associated with a single tuple space

$$H_j = \frac{\sum_{i=1}^{k} H_{ij}}{\log_2 k} \tag{3}$$

where the division by $\log_2 k$ is introduced in order to obtain $0 \leq H_j \leq 1$. If we have t tuple spaces then the entropy of the system is

$$H = \frac{1}{t} \sum_{j=1}^{t} H_j \tag{4}$$

where the division by t is used to normalise H, so that $0 \leq H \leq 1$. Being t the number of tuple spaces then it also represents the number of agents: let each agent work at rate $H_j r$, and tr be the maximum rate allocated to the sorting task. If we want to adapt the working rates of agents we have to scale their rate

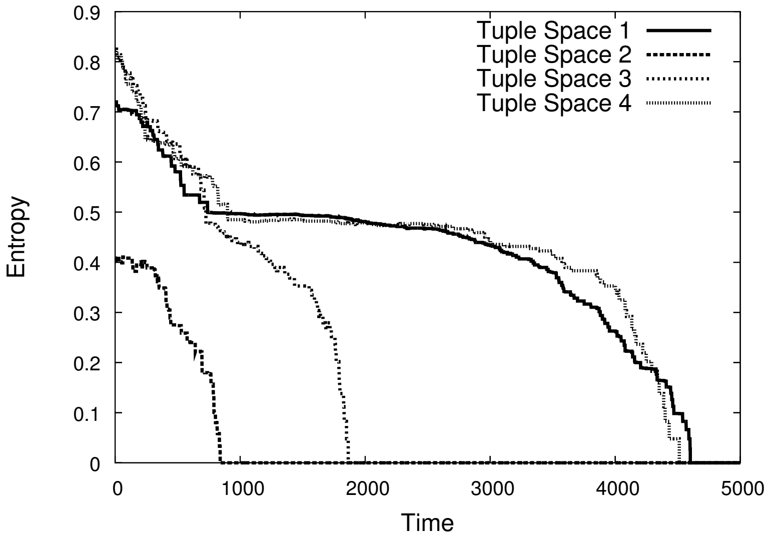

Fig. 5. Entropy of tuple spaces in the constant rate case: they all eventually reach 0, that is, complete ordering.

by the total system entropy, since we have that

$$\gamma \sum_{j=1}^{t} rH_j = tr \Rightarrow \gamma = \frac{t}{\sum_{j=1}^{t} H_j} = \frac{1}{H} \tag{5}$$

then each agent will work at rate $\frac{rH_j}{H}$ where H_j and H are computed periodically.

Using *load balancing* we introduced *dynamism* in our model: indeed in each simulation step the activity rate associated with a tuple space—i.e. the probability at a given step that an agent of the tuple space is working—is no longer fixed, but it depends on the entropy of the tuple space itself. Hence, as explained above, agents belonging to completely ordered tuple spaces can consider their goal as being achieved, and hence they no longer execute tasks. Moreover, this strategy guarantees a better efficiency in load balancing: agents working on tuple spaces with higher entropy, have a greater activity rate than others working on more ordered tuple spaces.

Using the collective sort specification with variable rates, we ran a simulation having the same initial state of the one in Section 4: the chart of Figure 6 shows the trend of the entropy of each tuple space. Comparing the chart with the one in Figure 5, we can observe that the entropies reach 0 faster than the case with constant rates: in fact, at step 3000 every entropy within the chart in Figure 6

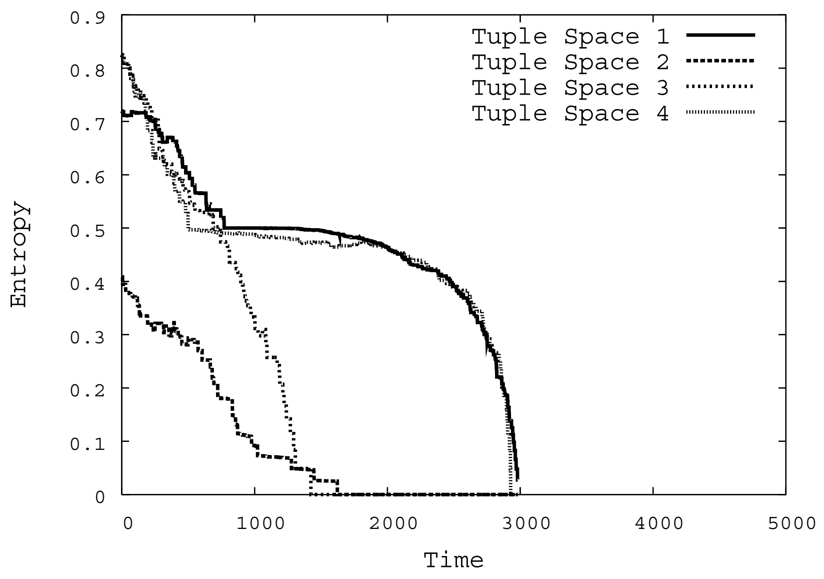

Fig. 6. Entropy of tuple spaces in the variable rate case: the system reaches the complete ordering since step 3000.

is 0, while with constant rates the same result is reached only after 4600 steps in Figure 5. The chart in Figure 7 compares the evolution of global entropy (see Equation 5) in the case of constant and variable rates: the trend of the two entropies represents a further proof that variable rates guarantee a faster stabilisation of the system, i.e. its complete ordering.

6 Related Works

Since the environment plays a crucial role in MASs, existing methodologies are incorporating guidelines for environment design. Furthermore, there exist a few methodologies also considering issues related to self-organization: we will briefly describe some of these methodologies and design practices.

In [14] it is recognised that traditional methodologies cannot help complex systems engineers: indeed, the required coordination between components is incompatible with the decomposition approach. We agree with that consideration and do not follow the decomposition approach, rather look for suitable solutions among the catalogue of known self-organising systems. Furthermore, in [14] some

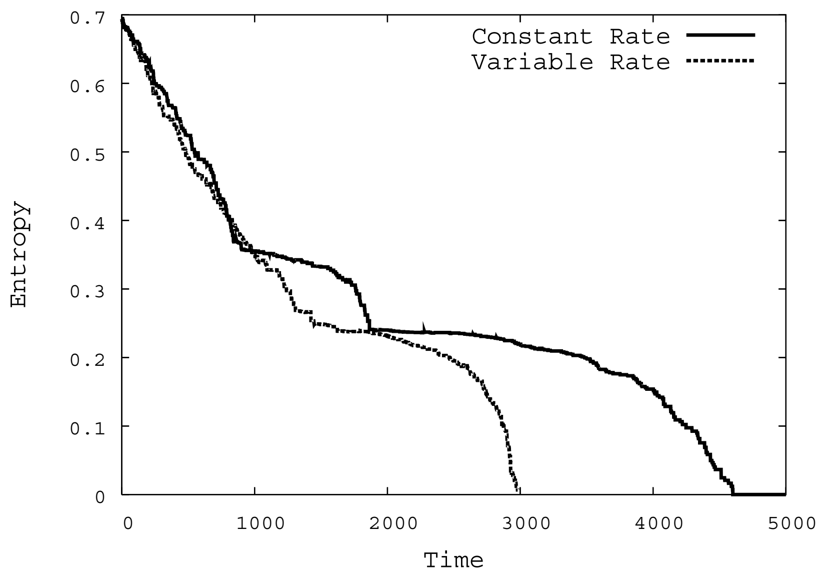

Fig. 7. Comparison of global entropy in the case of constant and variable rate: the latter reaches the complete ordering quicker.

useful guidelines are provided for the engineering process, although they are not organised in an actual methodology.

In [23] an architecture for MAS-based manufacturing control application is described. In particular, they describe an architecture and a case study which initial solution is inspired by stigmergy in ant colonies: although, they do not provide any hints on how to apply their approach to other domains or problems.

A more general approach is presented in [24] and focuses on reactive MAS for collective problem solving. A problem model is specified in the environment and constraints are assimilated to environment perturbations: agents collectively solve the problem by regulating environment perturbations. Hence, the MAS as a whole acts as a regulation process, in a way similar to automatic control. The methodology proposed is articulated in four steps, namely (i) defining the problem model, (ii) defining agent perceptions, (iii) defining agent interaction mechanisms and (iv) measuring the result as an emergent structure [24]. Although, methodology mainly focuses on problem solving suggesting that it is not well suited for environments as service providers. Instead, the main focus of our contribution is on developing a methodology for environments offering services with self-* capabilities, where environment is conceived as a run-time entity.

A more comprehensive methodology explicitly tailored for self-organising systems is ADELFE [13] which covers all the steps involved in adaptive systems engineering. Nonetheless, the authors overlooked the two most important issues in self-organization, that is (i) a general approach for devising a solution by emergence and (ii) metrics and performance assessment of the solution provided. Our approach, compared to ADELFE, is less comprehensive: composing them would however be an interesting future work.

7 Conclusion and Future Work

In this article we discuss an approach to the design of self-organising MAS environments: as far as methodological aspects are concerned, the approach relies on formal methods and tools for modelling and performing stochastic simulations. We briefly describe our A&A meta-model for MAS based on agents and artifacts, and discuss a basic architecture for self-organising environments: this solution features environmental agents as artifacts managers for the self-organisation process.

To better clarify our approach, we consider the case of a self-organising environment featuring automatic clustering of tuples of the same kind. The solution to this problem, that we call *collective sort*, has been derived from the swarm intelligence problem known as *brood sorting*, which we consider as a paradigmatic application of emergent coordination through the environment. Then, we elaborate on the idea of collective sort by evaluating entropy as a possible metric to drive the self-organisation process, showing its effectiveness.

Currently, we are working to refine the simulation framework developed on top of MAUDE, and to implement the collective sort application in TuCSoN. Future works include

- deeper testing of the convergence properties of the strategy;
- applying advancements in the field of formal methods, for the analysis of system specifications;
- identifying applicable patterns from self-organisation theory to the engineering of MAS environments.

References

1. Weyns, D., Omicini, A., Odell, J.: Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems **14**(1) (2007) 5–30 Special Issue on Environments for Multi-agent Systems.
2. Viroli, M., Holvoet, T., Ricci, A., Shelfthout, K., Zambonelli, F.: Infrastructures for the environment of multiagent systems. Autonomous Agents and Multi-Agent Systems **14**(1) (2007) 49–60 Special Issue on Environments for Multi-agent Systems.

3. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: Toward a theory of artefacts for MAS. Electronic Notes in Theoretical Computer Sciences **150**(3) (2006) 21–36 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.
4. Ricci, A., Viroli, M., Omicini, A.: *Construenda est* CArtAgO: Toward an infrastructure for artifacts in MAS. In Trappl, R., ed.: Cybernetics and Systems 2006. Volume 2., Vienna, Austria, Austrian Society for Cybernetic Studies (2006) 569–574 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), 5th International Symposium "From Agent Theory to Theory Implementation" (AT2AI-5). Proceedings.
5. Gardelli, L., Viroli, M., Omicini, A.: On the role of simulations in engineering self-organising MAS: The case of an intrusion detection system in TuCSoN. In Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F., eds.: Engineering Self-Organising Systems. Volume 3910 of LNAI. Springer (2006) 153–168 3rd International Workshop (ESOA 2005), Utrecht, The Netherlands, 26 July 2005. Revised Selected Papers.
6. Gardelli, L., Viroli, M., Omicini, A.: Exploring the dynamics of self-organising systems with stochastic π-calculus: Detecting abnormal behaviour in MAS. In Trappl, R., ed.: Cybernetics and Systems 2006. Volume 2., Vienna, Austria, Austrian Society for Cybernetic Studies (2006) 539–544 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), 5th International Symposium "From Agent Theory to Theory Implementation" (AT2AI-5). Proceedings.
7. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
8. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press (1999)
9. Casadei, M., Gardelli, L., Viroli, M.: Simulating emergent properties of coordination in Maude: the collective sorting case. In Canal, C., Viroli, M., eds.: 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR 2006, Bonn, Germany, University of Málaga, Spain (2006) Proceedings.
10. Camazine, S., Deneubourg, J.L., Franks, N.R., Sneyd, J., Theraulaz, G., Bonabeau, E.: Self-Organization in Biological Systems. Princeton Studies in Complexity. Princeton University Press (2001)
11. Zambonelli, F., Jennings, N.R., Wooldridge, M.J.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) **12**(3) (2003) 317–370
12. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A roadmap to artefacts. In Dikenelli, O., Gleizes, M.P., Ricci, A., eds.: Engineering Societies in the Agents World VI. Volume 3963 of LNAI. Springer (2006) 49–62 6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised, Selected & Invited Papers.
13. Bernon, C., Gleizes, M.P., Peyruqueou, S., Picard, G.: ADELFE: A methodology for adaptive multi-agent systems engineering. In Petta, P., Tolksdorf, R., Zambonelli, F., eds.: Engineering Societies in the Agents World III. Volume 2577 of LNAI. Springer (2003) 156–169 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17September2002. Revised Papers.
14. Bar-Yam, Y.: About engineering complex systems: Multiscale analysis and evolutionary engineering. In Brueckner, S.A., Serugendo, G.D.M., Karageorgos, A.,

Nagpal, R., eds.: Engineering Self-Organising Systems: Methodologies and Applications. Volume 3464 of LNCS. Springer (2005) 16–31 The Second International Workshop on Engineering Self-Organising Applications, ESOA'04, New York, USA, July 20, 2004, Selected Revised and Invited Papers.

15. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. Journal of Systems Architecture **52**(8–9) (2006) 443–460 Special Issue on Nature-Inspired Applications and Systems.
16. Phillips, A.: The Stochastic Pi Machine (SPiM) (2006) Version 0.042 available online at http://www.doc.ic.ac.uk/~anp/spim/.
17. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry **81**(25) (1977) 2340–2361
18. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic Pi-calculus. In Ingolfsdottir, A., Nielson, H.R., eds.: Workshop on Concurrent Models in Molecular Biology (BioConcur 2004), CONCUR 2004, London, UK (2004)
19. Ricci, A., Viroli, M., Omicini, A.: Programming MAS with artifacts. In Bordini, R.P., Dastani, M., Dix, J., El Fallah Seghrouchni, A., eds.: Programming Multi-Agent Systems. Volume 3862 of LNAI. Springer (2006) 206–221 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.
20. Gutowitz, H.: Complexity-seeking ants. In Deneubourg, J.L., Goss, S., Nicolis, G., eds.: 2nd European Conference on Artificial Life (ECAL'93), Brussels, Belgium (1993) 429–439
21. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Science of Computer Programming **41**(3) (2001) 277–294
22. Clavel, M., Duràn, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual. Department of Computer Science University of Illinois at Urbana-Champaign. (2005) Version 2.2.
23. Valckenaers, P., Holvoet, T.: The environment: An essential absraction for managing complexity in mas-based manufacturing control. In Weyns, D., Parunak, H.V., Michel, F., eds.: Environments for Multi-Agent Systems II. Volume 3830 of LNCS. Springer (2006) 205–217 Second International Workshop, E4MAS 2005, Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Papers.
24. Simonin, O., Gechter, F.: An environment-based methodology to design reactive multi-agent systems for problem solving. In Weyns, D., Parunak, H.V., Michel, F., eds.: Environments for Multi-Agent Systems II. Volume 3830 of LNCS. Springer (2006) 32–49 Second International Workshop, E4MAS 2005, Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Papers.