# Designing self-organising environments with agents and artefacts: a simulation-driven approach

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

Alma Mater Studiorum–Università di Bologna
Via Venezia 52, 47023 Cesena, Italy
Fax: +39 0547339208
E-mail: luca.gardelli@unibo.it
E-mail: mirko.viroli@unibo.it
E-mail: m.casadei@unibo.it
E-mail: andrea.omicini@unibo.it
*Corresponding author

**Abstract:** We propose a methodological approach for tackling the early design stages of self-organising Multiagent Systems (MASs). We adopt an architectural pattern based on the Agents and Artefacts (A&A) metamodel: self-organisation mechanisms are added to an existing environment of artefacts by embedding them into environmental agents. We rely on a three-stage design approach with modelling, simulation and tuning, so as to identify a suitable design of environmental agents and their interaction with artefacts. The main objective is to design a MAS environment providing services that self-organise in response to the unpredictable dynamics of the agents exploiting them.

As a case study, we analyse the problem called *collective sorting*, a service for decentralised sorting of items in MAS environments that was inspired by social insects' behaviour: the proposed solution features environmental agents and tuple spaces, whose design choices and evaluation have been driven by formal simulations.

**Keywords:** collective sorting; self-organisation; emergence; methodology; simulation; tuning; pattern; environment; Multiagent System; MAS; artefact.

**Reference** to this paper should be made as follows: Gardelli, L., Viroli, M., Casadei, M. and Omicini, A. (2008) 'Designing self-organising environments with agents and artefacts: a simulation-driven approach', *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No. 2, pp.171–195.

**Biographical notes:** Luca Gardelli studied Computer Science and Engineering at the Alma Mater Studiorum–Università di Bologna, Italy. Since 2005, he has been a PhD student in Computer Science in the Department of Electronics, Informatics and Systems (DEIS) at the Alma Mater.

Mirko Viroli has been an Associate Researcher since October 2002 at the Department of Electronics, Informatics and Systems (DEIS) of the Alma Mater Studiorum–Università di Bologna, Italy. He received his Laurea degree in Computer Engineering in October 1997 and his PhD in Computer Engineering in 2003, both from the Alma Mater. He has written over 100 articles on computational models and languages applied to coordination, agent-based systems, software engineering and programming languages, published in international journals, books, conferences and workshops.

Matteo Casadei received his Laurea degree in Computer Science Engineering in 2005 from the Alma Mater Studiorum–Università di Bologna, Italy. Currently, he is a PhD student in Computer Science Engineering at the Department of Electronics, Informatics and Systems (DEIS) of the Alma Mater.

Andrea Omicini is a Professor at the Department of Electronics, Informatics and Systems (DEIS) of the Alma Mater Studiorum–Università di Bologna, Italy. He received his Laurea degree in Electronic Engineering in February 1991 and his PhD in Computer Engineering in November 1995, both from the Alma Mater. He has written over 150 articles on coordination, software infrastructures, internet technologies, agent-based systems, artificial intelligence, software engineering and programming languages, published in international journals, books, conferences and workshops. He has edited 15 books and guest-edited 10 international journal special issues on agent-related issues.

## 1 Introduction

Self-organisation is increasingly being regarded as an effective approach to tackle the complexity of modern systems. This approach seems to be compelling owing to the possibility of developing systems exhibiting complex dynamics and adapting to environmental perturbations without requiring a complete knowledge of future surrounding conditions. The self-organisation approach promotes the development of simple entities that, by locally interacting with others sharing the same environment, collectively produce the target global patterns and dynamics by emergence. Many biological systems can be modelled using a self-organisation approach; well-known examples include food foraging in ant colonies, nest building in termite societies, the comb pattern in honeybees, brood sorting in ants (Bonabeau *et al.*, 1999; Camazine *et al.*, 2001). They have inspired the development of many artificial systems, such as decentralised coordination for automated guided vehicles (Weyns *et al.*, 2005; Sauter *et al.*, 2005), congestion avoidance in circuit-switched telecommunication networks (Steward and Appleby, 1994), manufacturing scheduling and control for vehicle painting (Cicirello and Smith, 2004) and self-organising peer-to-peer infrastructures (Babaoglu *et al.*, 2002). Furthermore, principles of self-organisation are currently investigated in several research projects that may have industrial relevance in the near future; notable examples include the SWARM-BOTS project (Mondada *et al.*, 2004), Amorphous Computing (Abelson *et al.*, 2000) and Autonomic Computing (Kephart and Chess, 2003).

However, the development of Self-organising Systems (SOSs) is driven by different principles with respect to traditional engineering. For instance, engineers typically design systems as a result of the composition of smaller elements, which are either software abstractions or physical devices, where composition rules depend on the reference paradigm (*e.g.*, the object-oriented one), and typically produce predictable results. Conversely, SOSs display nonlinear dynamics, which can hardly be captured by deterministic models and, though robust with respect to external perturbations, are quite sensitive to changes in inner working parameters. In particular, engineering a SOS poses two big challenges: How can we design the individual entities to produce the target global behaviour? And, can we provide guarantees of any sort about the emergence of

specific patterns? Even though the importance of these issues is generally acknowledged, few efforts have been devoted to the study of an engineering support both from methodologies and tools – except for a few explorations in the Multiagent System (MAS) community (De Wolf *et al.*, 2006; Bernon *et al.*, 2007).

In this article, we focus on methodological aspects concerning the early design stage of SOSs built by relying on the agent-oriented paradigm. With reference to the Agents and Artefacts (A&A) metamodel for MASs (Omicini *et al.*, 2006; Ricci *et al.*, 2006), we describe an architectural pattern that has been extracted from a recurrent solution in designing SOSs, as previously discussed in Gardelli *et al.* (2007b). This pattern is based on a MAS environment formed by artefacts (modelling nonproactive resources) and environmental agents (acting on artefacts so as to enable self-organising mechanisms). Then, we propose an approach for engineering systems that exploits this pattern and relies on simulation in the early design stage. In particular, the approach is articulated in three stages:

1 modelling

2 simulation

3 tuning.

In this approach, simulations of an abstract system model are used to drive design choices until the required quality properties are obtained, thus ensuring that the subsequent design steps actually lead to a correct implementation.

To clarify the approach, we apply it to a case study called *collective sorting*, a strategy for decentralised item sorting in MAS environments (Viroli *et al.*, 2007a). The environment is modelled as a flat set of tuple spaces as, for example, in Omicini and Zambonelli (1999), and the background sorting service is in charge of moving tuples until similar tuples are aggregated in the same space. Inspired by ants' brood sorting (Deneubourg *et al.*, 1991; Bonabeau *et al.*, 1999), we show how our three-stage approach can be exploited to produce a basic strategy aimed at improving its quality properties in terms of convergence to full sorting.

The remainder of the article is structured as follows: Section 2 describes the A&A metamodel and details the architectural pattern, providing motivations devised from self-organisation literature. Section 3 describes our simulation approach with respect to the architectural pattern. Section 4 exemplifies the concepts described in the paper through the case study of collective sorting. Finally in Section 5 we conclude and discuss future work.

## 2 Towards self-organising MAS environments

Agent-based computing is generally considered a good paradigm for accurately modelling SOSs. Indeed, the agent is a suitable abstraction for encapsulating the micro behaviours that eventually lead to self-organisation patterns at the social level. However, agents alone fail to capture those behaviours that are inherently distributed and non goal-oriented, but essential to many SOSs; consider as an example the process of pheromone evaporation, which is a fundamental brick of stigmergic systems (Bonabeau *et al.*, 1999; Camazine *et al.*, 2001). In particular, there is a class of processes that is

best modelled as a part of the environment rather than expressed in terms of agents. Accordingly, the environment is a fundamental concept for SOSs, and the method used for engineering MAS environments is therefore a key issue to inject SOS mechanisms into MASs.

### *2.1 The role of environment in self-organising systems*

From the analysis of natural SOSs (Bonabeau *et al.*, 1999; Camazine *et al.*, 2001; Solé and Bascompte, 2006) and experience in prototyping artificial ones (Weyns *et al.*, 2005; Sauter *et al.*, 2005; Casadei *et al.*, 2007; Gardelli *et al.*, 2007b; Mamei and Zambonelli, 2005), it is recognised that the environment plays a crucial role in the global SOS dynamics. A typical explanatory example is the case of stigmergy: as pointed out by Grassé (1959), among social insects, workers are driven in their activity by the environment. Indeed, in animal societies self-organisation is typically achieved by the interplay between individuals and the environment, such as the deposition of pheromone by ants or the movement of wooden chips by termites (Camazine *et al.*, 2001). In particular, these interactions are responsible for the establishment and sustainment of a feedback loop; in the case of ant colonies, positive feedback is provided by ants depositing pheromones, while negative feedback is provided by the environment through evaporation (Camazine *et al.*, 2001).

When moving to artificial systems, and to MASs in particular, there are a few questions that need to be answered. The first one is where to embed self-organising mechanisms. The above discussion promotes the distribution of concerns between active components and the environment – in the MAS context, between agents and the environment. This partially frees agents from the burden of system complexity, and provides a more natural mapping for those non-goal-oriented behaviours. The second question is how to find the minimum requirements for an environment to support self-organisation. From the definition of self-organisation provided by Camazine *et al.* (2001) we can identify some basic requirements:

- the environment should support indirect interactions among the components of a system
- the environment should support some notion of locality
- locality should affect interactions, *e.g.*, by promoting local ones.

Moreover, specific self-organising mechanisms may require an *active environment*, *i.e.*, the presence of active processes in the environment, making the environment evolve to a suitable state; *e.g.*, in pheromone-based systems, the environment may either provide a reactive *evaporation service*, or proactively act upon pheromone-like components to imitate the effect of evaporation.

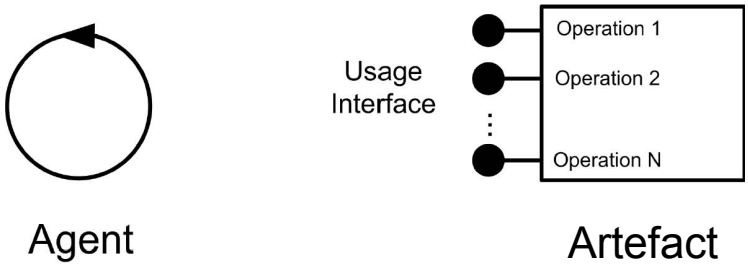
### *2.2 Engineering MAS environment: the A&A metamodel*

Software conceived according to the MAS paradigm is modelled as a composition of agents (autonomous entities situated in a computational or physical environment) that interact with each other and with environmental resources to achieve either individual or social goals (Weyns *et al.*, 2007; Viroli *et al.*, 2007b). Traditionally, the environment consists of a deployment context that provides communication services and access to

physical resources. In this context, MAS engineers design agents while the environment is just an output of the analysis stage. Recently, the environment has been recognised as an actual design dimension; then, MAS engineers can hide system complexity behind environmental services, freeing agents from specific responsibilities. In this article, we adopt the latter notion of environment, *i.e.*, the part of the MAS outside agents that engineers should design so as to reach the objectives of the application at hand.

In order to describe the environment, we have to provide suitable abstractions for environmental entities. As pointed out by Molesini *et al.* (2008), even though most of the current Agent-Oriented Software Engineering (AOSE) methodologies and metamodels provide little or no environment support, it is useful to adopt the A&A metamodel where a MAS is modelled by two fundamental abstractions: *agents* and *artefacts* (Omicini *et al.*, 2006; Ricci *et al.*, 2006). Agents are autonomous proactive entities encapsulating control and driven by their internal goal/task (Figure 1, left). When developing a MAS, sometimes entities require neither autonomy nor proactivity to be correctly characterised. This is typical of entities that serve as tools to provide specific functionalities. These entities are the so-called artefacts. Artefacts are passive, reactive entities providing services and functionalities to be exploited by agents through a *usage interface* (Figure 1, right). It is worth noting that artefacts typically realise those behaviours that cannot or do not require to be characterised as goal-oriented (Omicini *et al.*, 2006; Ricci *et al.*, 2006). Artefacts mediate agent interactions, support coordination in social activities and embody the portion of the environment that is designed and controlled to support MAS activities.

**Figure 1** A&A metamodel featuring agents as proactive goal-driven entities, and artefacts as encapsulating services to be exploited by agents through a usage interface



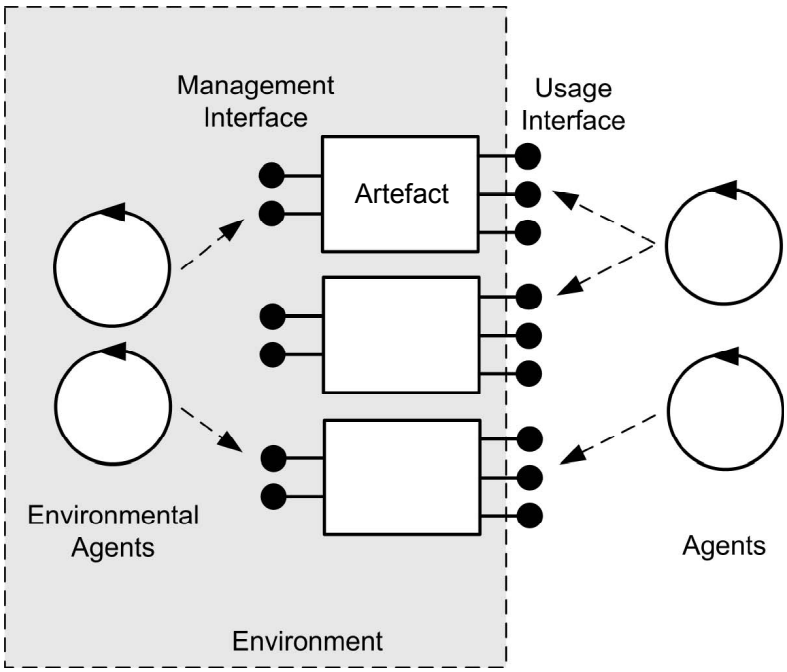

## *2.3 An architectural pattern for self-organising environments*

The components of modern computational systems often need to interact with an environment populated by *legacy systems.* Hence, the environment can be either completely or partially given. This is subject to investigation during the analysis phase (Molesini *et al.*, 2008). In the MAS context, during the design phase, resources are assigned to artefacts, providing a uniform way for agents to exploit resources. Unfortunately, in a scenario involving legacy systems, we may not have complete control over the environment, thus making it difficult to embed self-organising mechanisms within artefacts. Then, to inject self-* properties in MAS, we need to add a layer on top of existing environmental resources.

To this purpose, we rely on the notion of *environmental agent*: such agents are responsible for managing artefacts to achieve the target self-* property. Hence environmental agents are seen as distinct from standard agents (also called *user*

*agents*), which exploit artefact services to achieve individual and social goals. This recurrent solution has been encoded in the form of an architectural pattern (Gardelli *et al.*, 2007b) with reference to the A&A metamodel. As shown in Figure 2, environmental agents act upon artefacts through a management interface. This interface may be public, *i.e.*, accessible to all agents or, most likely, restricted and allowing access to operations typically granted to system administrators. A similar approach to achieve self-organisation, involving managers and managed entities, has been adopted in the Autonomic Computing community (Kephart and Chess, 2003).

**Figure 2** An architectural pattern for self-organising MAS featuring environmental agents responsible for sustaining feedback loop




Using this pattern produces several advantages. When working with legacy environmental resources – *e.g.*, provided by an existing infrastructure – relying on additional environmental agents is the only viable solution to add new properties and behaviour, owing to a limited control on environmental resources. When developing systems from scratch, the use of this pattern allows different mechanisms to be isolated, thus achieving a finer control on the overall system. Furthermore, we are able to identify and suppress conflicting dynamics that may arise when exploiting different self-organising mechanisms at the same time (Gardelli *et al.*, 2007b). It is worth noting that environmental agents differ from user agents because they play a special role in the system, bound within the environment as perceived by user agents. This is not in contradiction with previous works such as Omicini *et al.* (2006) and Ricci *et al.* (2006), but rather one step beyond; in fact, here user agents perceive the environment exactly in the same way, that is, populated only by artefacts, whereas environmental agents cannot interact with user agents, since they are somehow 'encapsulated' within the MAS environment.

This pattern can be successfully applied to embed self-organising mechanisms in MAS environments, especially to environmental services that do not natively support all the self-organisation features required. As an example we consider an environment populated by tuple spaces, provided by an existing coordination infrastructure as in Omicini and Zambonelli (1999), Mamei and Zambonelli (2005) and Weyns *et al.* (2005). In this context, agents are allowed to interact with a tuple space by using standard operations `rd`, `in` and `out` to read, remove and insert tuples. It could be interesting to enhance this environment with a background service for automatic tuple moving/changing mechanisms. This could be used to cluster similar tuples in the same space, diffuse tuples in the neighbourhood, or make them fade with a certain timing, and so on. Hence, according to the pattern, this problem can be tackled by designing environmental agents with the goal of performing the service in an effective and useful way, and transparently, to user agents. Studying a similar case is in fact the goal of Section 4.

## 3 Enhancing self-organising system design via simulation

### *3.1 Background and motivations*

Currently, the development of agent-oriented systems is supported by several software engineering methodologies. Most methodologies were initially conceived to cover specific issues, and then evolved to encompass the whole software process. For instance, the Gaia methodology (Zambonelli *et al.*, 2003) was mostly concerned with intra-agent problems, while the initial target of the SODA methodology (Molesini *et al.*, 2006) was to tackle the social, interagent dimension. On the other hand, other methodologies restricted the domain of applicability to a specific class of MAS, such as ADELFE for the Adaptive MAS theory (Bernon *et al.*, 2003).

Concerns like embedding self-organising mechanisms within an existing MAS, or engineering an SOSs from scratch, raise peculiar issues that are not typical or so crucial in current AOSE methodologies. For example, as stated in Section 2.2, while the environment plays a key role in self-organisation, just a few of the current AOSE methodologies provide explicit support for it (Molesini *et al.*, 2008). Furthermore, AOSE methodologies, as well as object-oriented ones, tend to focus on design-time aspects rather than run-time ones; in fact, it is common practice to assume that once a system has been designed, its structure will not change and will behave according to the specifications. The Autonomic Computing proposal suggests considering run-time issues at design-time; then, aspects such as maintenance become a functional requirement of the problem to be solved (Kephart and Chess, 2003), thus increasing the degree of autonomy and adaptiveness of the target system. Along this line, we promote the use of techniques that allow us to preview and analyse global system dynamics at design-time; indeed, when dealing with SOS, more attention should be devoted to observing the emergence of desired properties early in the design stage rather than waiting for the final implementation. In particular, when developing SOSs, we have to answer the following question: how can we design the individual environmental agent's behaviour in order to ensure the emergence of the desired properties? To tackle this issue, two approaches are typically exploited:

1 devising an *ad hoc* strategy by decomposition that will solve the specific problem

2 observing a system that achieves similar results, and trying to reverse-engineer its strategy.

It is generally acknowledged that the former approach is applicable only to a limited set of simple scenarios; owing to the nonlinearity in entity behaviours, global system dynamics become quite difficult to be predicted. Instead, in the self-organisation community, the latter approach is commonly regarded as more fruitful; in nature, it is possible to recognise patterns that are effectively applicable to artificial systems (Babaoglu *et al.*, 2006; De Wolf and Holvoet, 2007; Gardelli *et al.*, 2007b; Bonabeau *et al.*, 1999). Since it is quite unlikely to find a pattern that completely fits a given problem, it is common practice to rely on some modified and adjusted version. In the next section we will elaborate on the implications of these modifications.

Once a suitable strategy has been identified and adapted, how can we guarantee that it will behave as expected? Given the specifications of a SOSs, how to ensure the emergence of the desired global dynamics is still an open issue. While automatic verification of properties is typically a viable approach with deterministic models, verification becomes more difficult and soon intractable when moving to stochastic models. It is then useful to resort to a different approach, possibly mixing formal tools and empirical evaluations, so as to support the analysis of the behaviour and qualities of a design.

Before describing our approach in the next section, we would like to point out that it is not our goal to develop a brand-new complete methodology for MAS engineering. Instead, we would rather aim at integrating our approach within existing AOSE methodologies, and addressing the peculiar issues raised by self-organising MASs. For instance, by considering Gaia (Zambonelli *et al.*, 2003), our approach could be seen as a way to direct early design phases: on the one hand, this could help the developer in defining responsibilities for agents and services of the environment by taking inspiration from patterns found in natural systems; on the other hand, it could make it possible to preview the global dynamics of the MAS and tune its behaviour before committing to a specific design solution.

### *3.2 Designing environmental agents and their interactions*

As discussed in Section 2, in our approach environmental agents are used to embed self-organising mechanisms into a MAS environment. We assume that requirements have been collected and the analysis has been performed, in particular identifying the services to be performed by environmental agents; then our approach can be situated between the analysis and design phases, actually realising an *early-design phase.* In order to design environmental agents, we have first to provide a model for agents and environmental resources. We can then evaluate several strategies inspired by SOSs with the goal of discovering the one that could provide the quality attributes required for the application at hand. The model is analysed by simulation, then a precise environmental agents' behaviour and a set of working parameters are devised via a tuning process. In particular, we articulate this approach in three phases, possibly to be executed in a cyclic way: *Modelling*, to develop an abstract formal specification of the system; *Simulation*, to

use such a specification to qualitatively and quantitatively investigate the dynamics of the system; and *Tuning*, to change model parameters and behaviour so as to adjust system dynamics.

### *3.2.1 Modelling*

In the modelling phase we develop an abstract model of the system, providing a characterisation for:

- user agents
- artefacts
- environmental agents.

As far as artefacts are concerned, we can provide an accurate model of their behaviour with respect to the usage interface and set of services exposed – though it is often the case that a detailed description of the inner working is not available. Conversely, the repertoire of user agents' behaviour may be too vast to be accurately modelled. Indeed, in open environments, it is basically impossible to entirely foresee the future dynamics of agents – self-organisation is used precisely to adapt to unpredicted situations. Then, it is necessary to abstract from their peculiarity, resorting to probabilistic or stochastic models of user agents' behaviour. Models for these agents are developed in terms of usage of resources, *i.e.*, with respect to the observable behaviour and by abstracting away from inner processes such as planning and reasoning. The accuracy of the user-agent internal model is not so crucial, since self-organisation is built on top of indirect interactions mediated by the environment. Hence, it is sufficient to know how user agents perceive and modify their environment.

Once a suitable model for user agents and artefacts is provided, we move to the core part of modelling, that is, the characterisation of environmental agents. A suitable model for environmental agents is typically built on top of the services provided by artefacts, and functionally coupled with user agents' behaviour in order to establish and sustain a feedback loop. A feedback loop is necessary in every SOS. For example, in ant colonies, ants deposit pheromones which diffuse and evaporate in the environment; then, by the perception of the pheromone gradient, ants can coordinate their movements.

To find a candidate model for environmental agents, we can take inspiration from known SOSs and look for a model exhibiting or approximating the target dynamics. This step implies the existence of some sort of design-pattern catalogue, a required tool for an engineer of SOSs. Although this sort of catalogue does not exist yet, several efforts by different research groups are moving along this direction: several patterns with important applications in artificial systems have already been identified and characterised (Bonabeau *et al.*, 1999; Babaoglu *et al.*, 2006; De Wolf and Holvoet, 2007; Gardelli *et al.*, 2007b). Hence, even though patterns that perfectly match the target system dynamics can hardly be found, it is still feasible to identify some patterns approximating such dynamics; however, this typically requires changes of some sort. Given the chaotic behaviour that sometimes characterises SOSs, modifications should be done with care since they require expertise in mechanisms underlying SOSs. Moreover, modifications would also require a simulation-based approach as discussed in Section 3.2.2.

Although the model may be provided in several ways, we favour the use of formal languages; in fact, formal languages allow not only for further automatic analysis, but also for ambiguity avoidance and precise selection of model features. For instance, given a specification and a suitable set of parameters, it is possible to analyse the system dynamics by means of simulation and formally verifying system properties. To this purpose, tools like PRISM-Probabilistic Symbolic Model Checker (Hinton *et al.*, 2006) or SPiM-Stochastic Pi Machine (Phillips and Cardelli, 2004) provide a satisfactory support; though such modelling languages are not explicitly tailored to MASs, they usually provide abstractions – such as processes or modules – that can somehow be assimilated to an agent.

### *3.2.2 Simulation*

Given a formal specification of the system to investigate, before performing simulations to investigate system dynamics it is necessary to provide a suitable set of parameters for the model. The type of parameter depends on the kind of simulation we are interested in. However, since when dealing with self-organising MASs we mostly rely on stochastic simulation (capturing timing and probability aspects), such parameters are typically expressed in terms of *rates of action*, defined according to suitable statistical distributions – typically, an exponential distribution is used owing to the *memoryless property* (*i.e.*, to generate new events it is not necessary to know the whole event history, but only the current state), as in Continuous-Time Markov Chains (Kulkarni, 1995).

Hence, the main concern in the simulation phase is to devise proper system behaviours and valid ranges for the system parameters (action rates): we do not rely on any specific guideline for devising those parameters even though we recognise they should reflect the conditions in the deployment scenario, otherwise the simulations results would be meaningless. While parameters for artefacts can be accurately measured, user agents provide a major challenge since we cannot foresee all their possible behaviours. Hence, we have to make assumptions about artefact behaviours both from the qualitative (*e.g.*, rational exploitation of resources) and quantitative (*e.g.*, rate of actions and rate of arrivals/departures) standpoints.

Once the parameters for artefacts and agents are available, we devise an initial set of parameters for environmental agents and perform standard statistical analysis upon simulation results. Then the parameters of environmental agents need to be tuned until the desired global dynamics are observed. At this stage, it is common to test the global system behaviour in different environmental conditions that are representative of expected or actual scenarios.

### *3.2.3 Tuning*

In the *tuning* phase, environmental agents' behaviour and working parameters are successively tuned until the desired dynamics are observed. It is worth noting that setting parameters to arbitrary values may lead to unrealistic scenarios. Furthermore, the working rate of environmental agents may affect the actual working rate of artefacts: in a realistic scenario, computational resources are typically limited; hence, increasing the working rate of environmental agents may require a decrease in the service rate of artefacts. Without considering this problem, the dynamics of the deployed system may significantly deviate from the expected ones.

Once the desired dynamics have been previewed, the set of parameters works as a coarse characterisation for an actual implementation. At the end of the tuning process, we may realise that the devised set of parameters does not satisfy performance expectations, features unrealistic values with respect to the execution environment, or deviates the system from the desired behaviour. In any of these scenarios, we cannot proceed to the actual design phase since the system is not likely to behave properly when deployed. Hence, backtracking the modelling choices and evaluating other modifications or approaches is required. Conversely, when a model meets the target dynamics, and the parameters lie within the allowed ranges, we can proceed by providing a more accurate statistical characterisation of system behaviours. However, since stochastic simulation provides a partial view of the possible system dynamics, accurate predictions about the system behaviour cannot be provided; *i.e.*, the emergence of a given property is observed with a frequency that approximates the actual probability. Critical systems may hence require additional in-depth formal analysis.

### *3.3 Applicability of the approach*

As previously stated, our approach is not to be considered as a complete methodology, but as a set of best practices devised from experience in modelling and prototyping artificial SOSs. Our goal is to introduce scientific analysis techniques and tools in a systematic manner to conduct the early design stage of MAS engineering. The practice of *simulating before deploying* is more common in fields where system deployment is much more complicated/expensive, or involves physical devices, *e.g.*, in the case of collective robotics. Conversely, the use of simulation tools – even if well known – is not common practice in mainstream software engineering, despite the many advantages especially in engineering complex systems. A limitation of the approach is the dependence on the existence of a natural system that displays the desired, or similar, dynamics. However, to our knowledge no alternative exists, *i.e.*, forward engineering is not feasible. Furthermore, the investigation of natural systems has already made it possible to solve several computer science problems (see Bonabeau *et al.* (1999) for a comprehensive discussion).

As far as applicability is concerned, we suggest the use of this approach, along with architectural patterns, for every system featuring complex dynamics or self-organising mechanisms which can be adequately captured by the A&A metamodel. So, our approach is applicable to many distributed systems, including those not initially conceived as a MAS. Furthermore, the approach could be fruitfully applied to physical MASs, as in the case of collective robotics scenarios.

In the context of software systems, we consider this approach particularly suitable for the coordination of large-scale distributed systems such as in stigmergy and pervasive computing. For instance, Linda-like tuple spaces provide a coordination medium that can be easily mapped to artefacts. There exist various incarnations of Linda in the shape of coordination middleware for MASs such as TOTA (Mamei and Zambonelli, 2005), TuCSoN (Omicini and Zambonelli, 1999) and SwarmLinda (Menezes and Tolksdorf, 2003). Some of them, such as TOTA and SwarmLinda, already embed some mechanisms to support self-organisation. Within an environment populated by tuple spaces, interaction between agents and artefacts is reified by tuples. Hence, the role of environmental agents is to perturb the environment by properly moving tuples or manipulating them. In the first case, the objective is to make the same configuration

patterns emerge, *e.g.*, grouping similar tuples as done in the example in the next section. In the second case, annotations such as timestamps and counters may be linked to tuples, supporting processes that require a partial history of events, *e.g.*, pheromone evaporation.

## 4 The case of collective sorting

In this section we analyse the case study known as *collective sorting* (Viroli *et al.*, 2007a) in order to exemplify our methodology for the early design of self-organising MAS environments. We refer to an environment where artefacts have the shape of tuple spaces and agents are allowed to insert and retrieve tuples on a content-based approach. This is a typical scenario of agent-mediated interaction, general enough to support a wide range of applications, from complex workflow engines (Ricci *et al.*, 2002) to stigmergic MASs (Weyns *et al.*, 2005; 2007; Sauter *et al.*, 2005).

A typical problem of environments based on tuple spaces is that it is generally difficult to retrieve the tuples of interest when these can be inserted in any tuple space of the net. A possible solution is to allow agents to find tuples on a similarity basis, *e.g.*, by grouping similar tuples in the same tuple space and separating different tuples. As a consequence, if a tuple is found in a tuple space, similar ones are likely to be found later in the same space. Furthermore, an ordered environment allows for a better batch processing of 'items', *e.g.*, for applying aggregation techniques, checking consistency, and so on.

Collective sorting amounts to providing an environment that offers a 'background' sorting service. Given a set of $N$ tuple spaces and a statically defined clustering of tuples into $N$ *kinds*, collective sorting is about moving tuples towards the fully sorted situation where each space hosts only tuples of the same kind. Accordingly, sorting should proceed in dynamic and unpredictable ways, in which user agents keep interacting with tuple spaces, that is, moving, inserting and dropping tuples. Therefore, the tuple space that will eventually aggregate a certain kind is not to be decided *a priori*; rather it should be implicitly and probabilistically selected as tuples start aggregating in one space rather than another owing to the effect of multiple tuple movements – namely, in a self-organising style. This approach is meant to tackle robustness, which is regarded here as more important than performance: we need sorting to be a property that eventually emerges in spite of external interactions. Of course, the more user agents keep altering tuple configurations, the more resources should be devoted to sorting in order to achieve convergence.

In conformity to the architecture described in Section 2, the sorting task is hence assigned to a set of environmental agents, whose goal is to keep the environment ordered as much as possible. In this section we apply the proposed approach to provide a sound model of the behaviour of such agents. Taking inspiration from a similar problem – *brood sorting* in ant colonies – in Step 1 we identify a possible model for the behaviour of environmental agents. In Step 2 we discuss the outcomes of several simulations of system behaviour, showing that the solution is not completely adequate, for it does not always lead to full sorting. In Step 3 we then tune the system model by introducing a mechanism resembling *simulated annealing*. Further simulations show the adequacy of the new model, and emphasise the behaviour of sorting during user-agent interactions.

### *4.1 Identifying a suitable approach in nature*

Collective sorting in distributed tuple spaces is reminiscent of a classical problem in robotics known as *segregation*, in which robots roam the ground with the goal of finding, grouping and separating items.

Solutions to this problem are typically searched for in nature, which is a rich source of simple but robust strategies. Segregation has already been manifested by social insects in *brood sorting* (Bonabeau *et al.*, 1999). When organising the surrounding physical environment, ants need to group and to keep broods and larvae separate from each other. Although ant behaviour is still not fully understood, there are several models able to mimic the dynamics of the system. Ants wander randomly and feature a behaviour modelled by two probabilities: the probability to pick up ($P_p$) and drop ($P_d$) an item. The idea is that an ant:

- picks up an item if its concentration is lower than the one measured in previous experience
- starts wandering randomly
- drops the item in a place featuring a concentration higher than where the item was picked up.

The ant-based solution to brood sorting is intrinsically self-organising; in fact, ants are guided by spatially local observations and motivated only by the need for picking items up where concentration is low, and dropping them where concentration is high. Such numerous interactions make full sorting (*i.e.*, the segregation pattern) emerge at the global level. Sorting performance is suboptimal – *i.e.*, noncomparable with solutions based on global observations – but intrinsically robust; indeed, it is able to promptly react to changes in the environment (*e.g.*, new brood, larvae, or ants are dynamically added or removed), faults like an environment split (*e.g.*, a barrier splitting the ground in two parts) and local malfunctions (*e.g.*, ants behaving in a completely different way). As a consequence, it is interesting to seek a solution to collective sorting by taking inspiration from the ant-based solution to brood sorting.

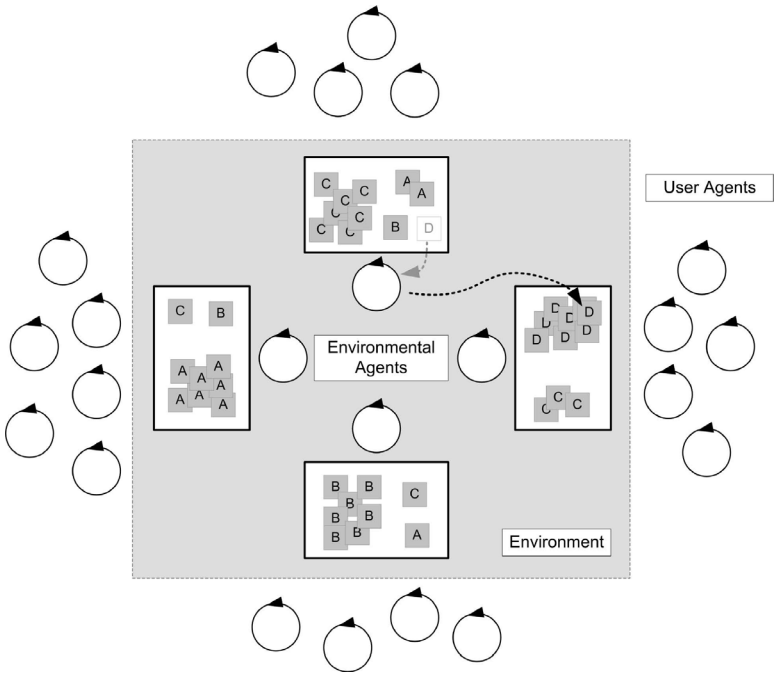
However, as already discussed, the above solution needs to be significantly adapted since the application scenarios of brood sorting and collective sorting feature key differences. First of all, our scenario is not a continuous environment, featuring instead a set of $N$ tuple spaces, each of them being a concentrated, conceptually unlimited bag of tuples. Secondly, our environmental agents are not likely to move and carry tuples. Rather, for obvious performance reasons, these agents should reside in one of the $N$ sites and send tuples away when needed. Finally, instead of perceiving items on a locality basis, environmental agents should be able to look for tuples in either a local or a remote tuple space – with the latter operation obviously being more expensive.

### *4.2 Step 1: Modelling collective sorting*

We consider $N$ environmental agents, also called *sorting agents*, each situated in a different site hosting one of the $N$ tuple spaces. Each agent is hence assigned to one tuple space, which can be regarded as a tuple space local to the agent. Interactions with this tuple space are less expensive than interactions with other tuple spaces. Similarly to ants, an agent can perform only partial observations on the system, *i.e.*, observations on

the local tuple space (where the item may be picked up) and observations on a remote tuple space chosen randomly (where the item may be dropped). According to such observations, if it can be inferred that a tuple should be removed from the local tuple space, the agent will locally remove the tuple and move it to a remote tuple space. This observation-action cycle is executed by adopting a fixed sorting-agent rate $r$, and a global sorting-agent rate which is equal to $N * r$ – basically the number of moving attempts per time unit. This scenario is depicted in Figure 3.

**Figure 3** Basic architecture for collective sorting




Therefore, each agent features the goal of moving tuples away from its local tuple space whenever they do not form a collection. The protocol of each agent can be described as follows:

- A remote tuple space $R$ is drawn randomly.
- A 'uniform `rd`' operation is performed on the local tuple space ($L$), yielding a tuple of kind $K_L$.
- A 'uniform `rd`' operation is performed on $R$, yielding a tuple of kind $K_R$.
- If $K_L \neq K_R$, a tuple of kind $K_R$ is moved from $L$ (if any exists there) to $R$.

The uniform `rd` operation, also called `urd`, is an operation by which an agent can read any tuple from a tuple space, *i.e.*, every tuple features the same probability of being retrieved. If an execution of `urd` on a tuple space yields a tuple of kind $K$, that tuple space likely has a high concentration of tuples of kind $K$. Consequently, $K$ is supposed to be a good aggregator for the tuple space. After executing the third task, the agent knows that space $L$ is an aggregator for $K_L$ tuples and $R$ an aggregator for $K_R$ tuples.

Accordingly, the rationale behind the fourth task is that if $K_R$ and $K_L$ are different, the agent can fruitfully send a tuple of kind $K_R$ from $L$ to $R$, so that both $K_R$ in $R$ and $K_L$ in $L$ become stronger aggregators.

This first solution can be turned into any stochastic specification language (Hinton *et al.*, 2006; Phillips and Cardelli, 2004). We relied on the stochastic simulation library for the MAUDE term-rewriting language discussed in Casadei *et al.* (2007), though any other language could be used.

### *4.3 Step 2: Simulating collective sorting*

The observations and the decisions taken by the agent are affected by probability, so the correctness of this distributed algorithm needs to be checked by simulation, in order to verify whether it is possible to achieve complete ordering from any initial situation, even the most chaotic one. As an example, consider an initial chaotic configuration where every tuple space features the same concentration of tuples for each kind. Supposing $N = 4$, the corresponding system state can be represented by the syntax:

T1[25,25,25,25],T2[25,25,25,25],T3[25,25,25,25],T4[25,25,25,25]

expressing the state where each tuple space Ti features 25 tuples per kind (the different kinds labelled K1, K2, K3 and K4). An example of a simulation trace is pictorially represented in Figure 4(a), which reports the dynamics of the 'winning' tuple in each tuple space, namely, the tuple that eventually aggregates there. Note that tuples reach their full aggregation level at different instants, and mostly in unpredictable ways. As a further example, the chart in Figure 4(b) shows the temporal evolution of tuple space T1: notice that only tuples of kind K1 aggregate there even though the initial concentration was the same for all kinds. For instance, around step 1000 it is easy to recognise a bifurcation promoting aggregation of tuples of kind K1 instead of K2.

It is also interesting to analyse the trend of the entropy for each tuple space as a way to estimate the degree of order in the system through a single value: since the simulated strategy tries to increase the inner order of the system, we expect the entropy to decrease to zero as shown in Figure 4(c). The entropy associated with a tuple space is computed in the standard way (Casadei *et al.*, 2007), by considering the concentration for each single tuple kind and normalising the total entropy in a range between 0 and 1. Each chart reports the number of protocol instances (moving attempts) executed by agents. Supposing a single-agent rate of 0.25, the global agent rate is 1.0, with an average of one simulation step per time unit – meaning that full sorting is reached after 3000 time units. Other simulations performed with a different number of tuples and tuple spaces show similar qualitative results.

In general, the outcome of a simulation should highlight the system performance, but it can sometimes show flaws in the design. In our case, though it first appeared that the proposed model always leads to complete sorting from any initial configuration of tuples, more thorough simulations showed that there are certain stable states attracting the system trajectory and having positive entropy, that is, characterised by a noncomplete degree of sorting. A state of this kind is called *local minimum* (for entropy). An example of such a minimum is the following state, obtained by the traces shown in Figure 5(a) and Figure 5(b):

T1[100,0,0,0],T2[0,69,0,0],T3[0,31,0,0]),T4[0,0,100,100]

**Figure 4** Charts of a simulation trace: (a) winning tuple; (b) tuple space T1; (c) entropy in each tuple space

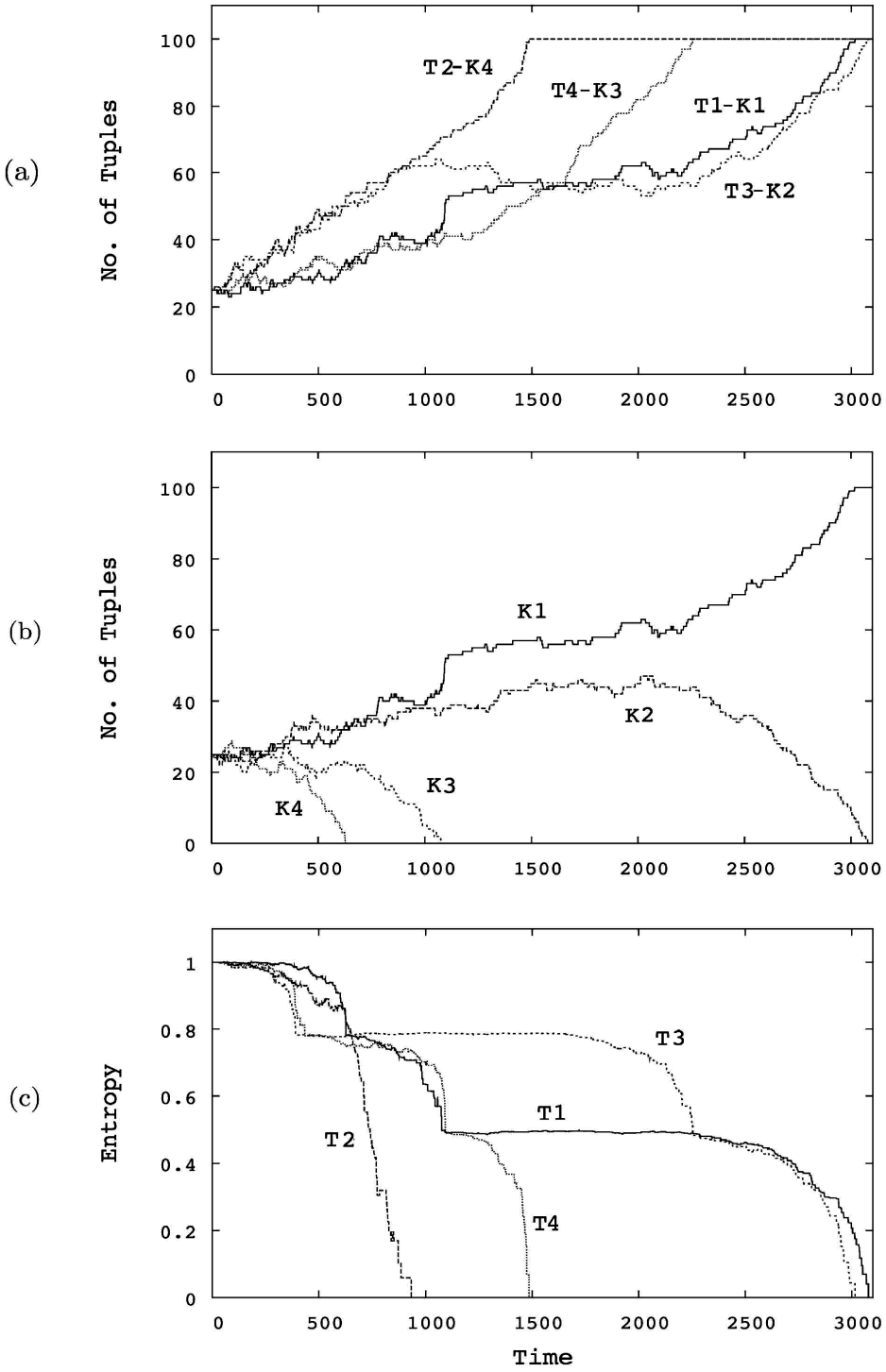
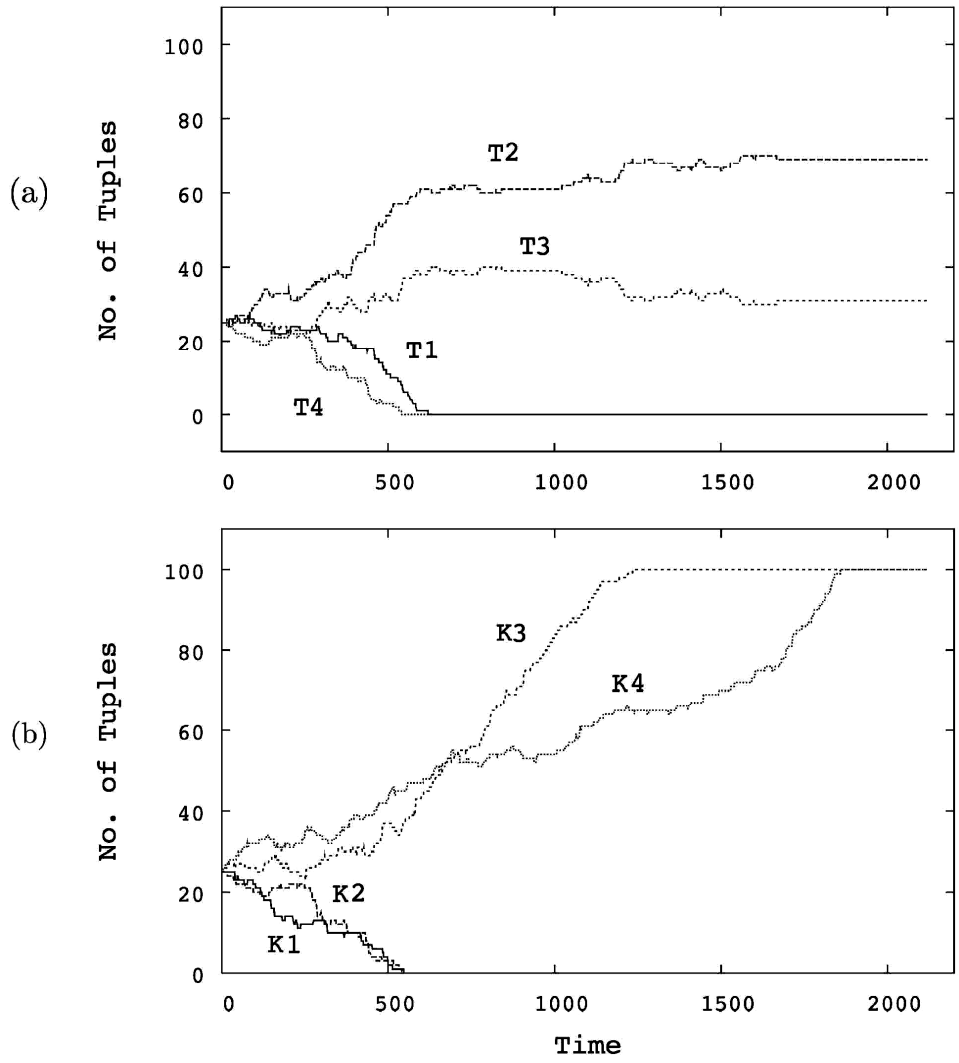


**Figure 5** Charts of a simulation trace to a local minimum: (a) tuple kind K2 aggregating in spaces T2 and T3; (b) both kinds K3 and K4 aggregating in space T4




Tuple kind K2 is the only one aggregating in both spaces T2 and T3; at the same time, kinds K3 and K4 both aggregate in space T4. It is easy to recognise that once this local minimum state is reached, agents will not move tuples any longer since it is not possible to find a space where a tuple aggregates less than elsewhere. Such a state can then be considered an attractor; in fact, simulations starting from states sufficiently near to them appear to converge back to this local minimum. This makes the described strategy inadequate, so a tuning of the model is required in order to find a complete solution.

### *4.4 Step 3: Tuning collective sorting*

The local minimum analysed above cannot be escaped owing to the fact that the developed strategy does not explicitly avoid the case in which the same tuple kind aggregates in two different tuple spaces. Indeed, owing to the fourth step in the agent's protocol, nothing is done when $K_L = K_R$. Hence, the same tuple kind may fully aggregate on two different tuple spaces, leading to the complete aggregation of the two remaining tuple kinds in a single tuple space as shown in the above local minimum.

These two issues can be solved by more carefully analysing the brood sorting problem for social insects. There, an ant takes an item and releases it when a new place featuring a greater concentration of items is found. The concentration is expressed as quantity of brood over unit of space. Implicitly the ant is able to compare the amount of brood to the standard quantity represented by the amount of vacuum.

To implement a mechanism supporting this idea, we add another kind of tuple, called `noise`, and initially suppose it to be constant throughout sorting. Now an observation by a `urd` can be 'perturbed', yielding a noise tuple. As in the previous model, if the local and remote observations are different, a tuple is moved from the local space to the remote one. Moreover, if an observation is perturbed by the reading of a noise tuple, the correctness of moving is now probabilistically altered. However, the probability of picking a tuple in T3 is expected to be higher than in T2. This should promote tuples of kind K2 to leave T3 more quickly. As a result, this mechanism is expected to globally result in complete sorting.

This mechanism resembles the concept of *simulated annealing* (Kirkpatrick *et al.*, 1983), where a perturbation is added to an optimisation algorithm in order to avoid the risk of finding a nonoptimal solution. Such a perturbation is initially high and continuously fades as the system finds new solutions, until it completely disappears.

In our case, the occurrence of noise tuples models such a perturbation. What then should the dynamics of noise through time be? A possibility would be to initially set an equal amount of noise in every tuple space, either leaving it unaltered during a system life cycle, or decreasing it at a fixed rate. However, this choice would require setting the amount of noise at design-time, while an optimal value depends on the average occupation of tuple spaces during system execution (Viroli *et al.*, 2007a). This situation is not appealing since we want our approach to work independently of the number of tuples in the system. What we are actually looking for is a fully adaptive noise mechanism, where an initially very low concentration of noise increases as the system approaches a local minimum, and decreases if the minimum is escaped. In this way, we could expect the system performance to be only slightly affected if the system stays sufficiently far from local minima, and on the other hand, the noise production may become significant only in unfortunate cases where local minima are approached.

To achieve this result, we will manage noise as follows:

- initially, every tuple space features a concentration of only one noise tuple.
- whenever two tuple spaces seem to aggregate the same tuple kind, noise is increased.
- when some tuple is correctly transferred – without an observation perturbed by noise – noise is decreased.

Accordingly, we change the environmental agent design by relying on the following protocol:

1 A remote tuple space $R$ is drawn randomly.

2 A uniform `rd` operation is performed on $L$, yielding a tuple of kind $K_L$.

3 A uniform `rd` operation is performed on $R$, yielding a tuple of kind $K_R$.

4 If $K_L \neq K_R \neq$ `noise`, a tuple of kind $K_R$ is moved from $L$ to $R$.

5 If $K_L \neq K_R =$ `noise`, a tuple of kind $K_L$ is moved from $L$ to $R$.

6 If `noise` $\neq K_R = K_L$, noise is increased by 1 in $L$.

7 If `noise` $\neq K_L \neq K_R \neq$ `noise`, noise is decreased by 1 in $L$.

Now, this protocol allows the situation where both $K_L$ and $K_R$ are noise. The fourth and fifth tasks say that different observations in $L$ and $R$ should always cause transfer: if $K_R$ is not noise, a $K_R$ tuple is moved to $R$, otherwise a $K_L$ tuple is moved to $R$. The sixth task increases noise if $L$ and $R$ are aggregating (nonnoise) tuples of the same kind ($K_R = K_L$), and finally the seventh task decreases noise if a nonperturbed transfer is executed.

Considering now the worst case of a symmetric local minimum:

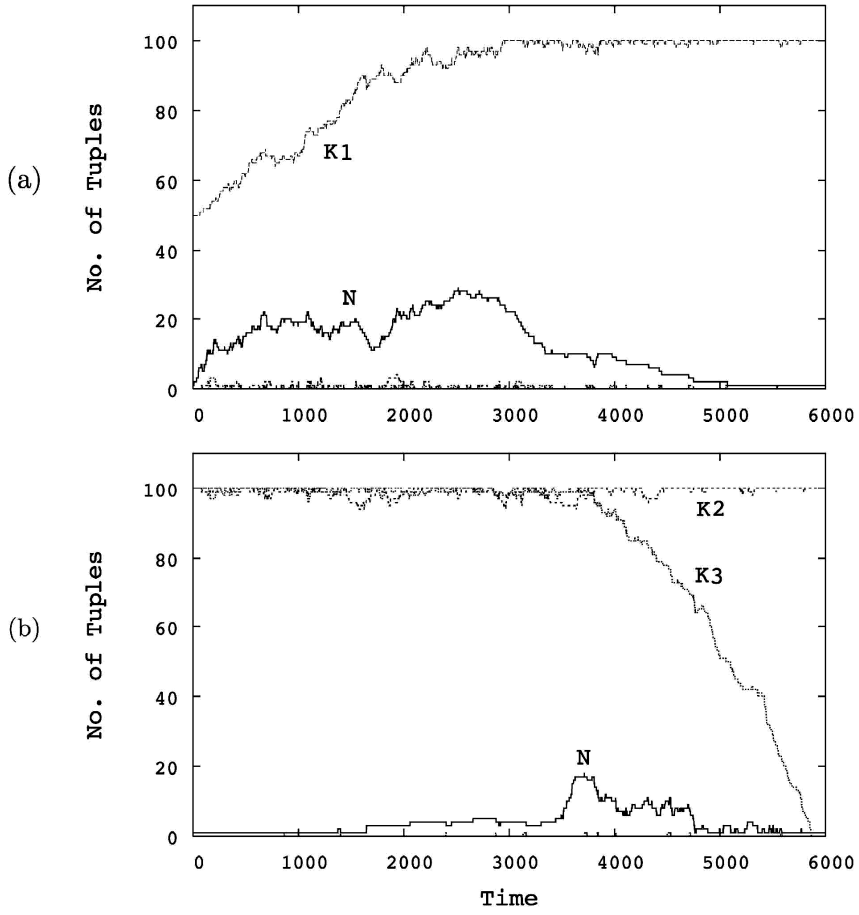
T1[100,100,0,0],T2[0,0,50,0],T3[0,0,50,0],T4[0,0,0,100],

we expect noise to start increasing both in tuple space T2 and tuple space T3. At some point, movement of K3 tuples between T2 and T3 occurs since some noise is observed. Owing to a bifurcation effect, if either space T2 or T3 features a concentration of K3 tuples greater than noise, more K3 tuples are transferred there, making that space eventually aggregate K3 tuples. Accordingly, the other tuple space is emptied, loses noise tuples and becomes the target of tuples of kind K1 and/or K2. This is actually what can be observed from the traces in Figure 6(a) and Figure 6(b), showing how the local minimum is escaped in spaces T2 and T1. In both cases we can see that, as noise tuples increase, the system escapes the local minimum configuration, leading to the fading of noise tuples as a consequence.

More simulations performed on this solution show that:

- using noise slightly affects performance, for systems typically stay away from local minima and generate little noise
- starting from a local minimum, the system is always able to escape the local minimum
- full sorting is always reached
- these results are independent of the number $N$ of tuple spaces (and kinds).

**Figure 6** Charts of a simulation trace escaping from a local minimum: (a) situation in space T2: winning tuple and noise in evidence; (b) situation in space T1: kind K3 leaves the space




## *4.5 Evaluation of reactiveness*

Having found a promising solution, it is interesting to get back to simulation. In this section we report the final results, and evaluate interesting system parameters to be used in subsequent steps of MAS design.

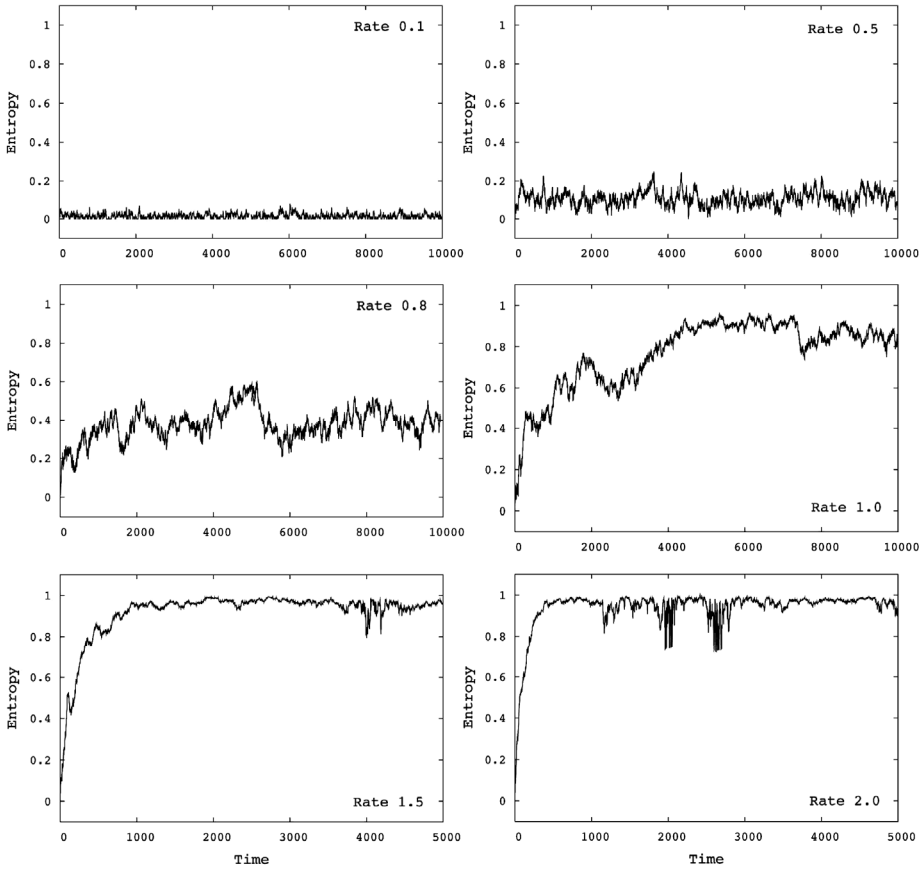
The main purpose of solving the collective sorting problem for tuple spaces using a self-organising approach is to tackle unpredictable interactions with the environment.

The typical usage scenario includes user agents exploiting the coordination service provided by tuple spaces, and inserting and removing tuples. The details of this behaviour cannot be known *a priori*, so sorting should be able to react to changes in the surrounding conditions in a fully adaptive way. This section shows how the ratio between user agent rate and sorting agent rate, called *perturbation/sorting ratio*, influences the result of sorting. To this end, we keep the global sorting rate fixed to 1.0 and include a *change rate* for user agents in the simulation, that is, the rate by which a user agent randomly moves a tuple from one space to another. Starting from an initially sorted configuration of tuples (400 tuples, $N = 4$) and depending on the change rate, we can easily expect that:

- full sorting is almost always maintained
- a certain level of (partial) sorting can be maintained
- the system becomes more and more unsorted as time passes.

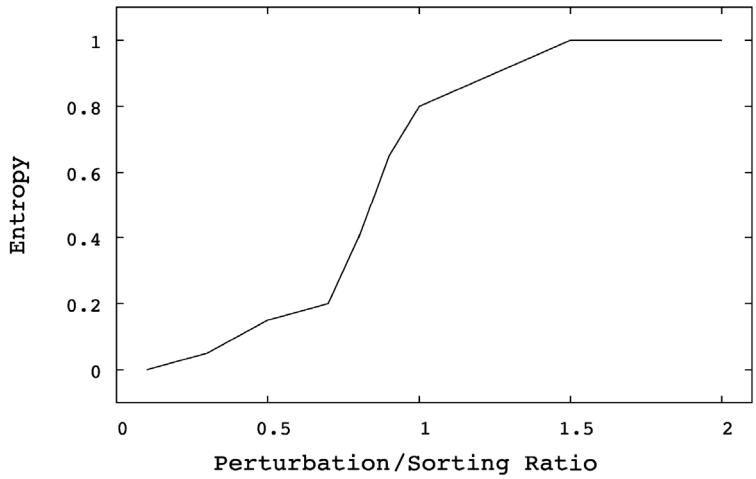
The evolution through such situations is reported in Figure 7, where each chart shows the evolution of entropy over time for different rates.

**Figure 7** Evolution of entropy with different perturbation/sorting ratio




As shown in Figure 8, the key factor is the perturbation/sorting ratio, which gives a clear indication of the adequacy of sorting resources, in terms of the maximum guaranteed level of entropy. Along with the identified environmental agent behaviour, the bound of the perturbation/sorting ratio, set to 0.5, is a critical system parameter that can be revealed only by simulation and then may be exploited in subsequent design steps (not discussed here). For instance, a form of load-balancing is required to make sure that the resources of sorting are adequate and self-adaptable to the current degree of disorder, *i.e.*, they increase when needed and then decrease. Techniques related to the prey-predator approach as studied, for example, by Gardelli *et al.* (2006; 2007a) could be evaluated in the subsequent steps of design.

**Figure 8** Maximum entropy depending on perturbation/sorting ratio




## 5 Conclusion

In this article we discussed an approach to drive the early-design phase in the engineering of self-organising MASs. In particular, the approach is based on the A&A metamodel, which describes a MAS in terms of agents and artefacts. With respect to the metamodel, we introduced the role of environmental agents in the form of an architectural pattern: such agents are responsible for those environmental processes/behaviours needed to close the feedback loop together with other agents. Hence, in our approach, designing a self-organising MAS consists in properly designing environmental agents.

Since the dynamics of SOSs tend to be complex, we resort to stochastic simulations in order to design environmental agents. In particular, the approach is articulated in three steps: modelling, simulation and tuning. The advantages of using simulation in the software engineering process are typically overlooked; recently, however, simulation has been gaining consensus in the MAS community for tackling complex-system design (De Wolf *et al.*, 2006; Bernon *et al.*, 2007; Fortino *et al.*, 2006; Uhrmacher, 2002).

As a case for applying this method, we considered the problem of decentralised sorting in environments built upon tuple spaces: specifically, the proposed solution clusters similar tuples in the same space while separating different tuples. The solution to this problem, called *collective sorting*, has been initially inspired by the swarm intelligence problem known as *brood sorting* (Deneubourg *et al.*, 1991; Bonabeau *et al.*, 1999). To improve the algorithm convergence, we introduced noise tuples that allow local minima to be avoided by perturbing the tuple-space state. This approach conceptually resembles simulated annealing.

Through this case study, we can argue that SOSs can hardly be designed without resorting to simulation-driven approaches from the early-design phase. Furthermore, the use of environmental agents is seemingly unavoidable when using existing infrastructures, such as tuple spaces, and promotes a neat separation between environmental services and self-organising mechanisms.

We also identified two main future research developments:

1 the integration between the presented approach and existing AOSE methodologies

2 the adoption of formal-analysis techniques and model-checking tools for validating simulation results, and for characterising target system behaviour in a more accurate way.

## References

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas, F., Knight, J., *et al.* (2000) 'Amorphous computing', *Communications of the ACM*, Vol. 43, No. 5, pp.74–82.

Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G.A., Ducatelle, F., Gambardella, L.M., Ganguly, N., *et al.* (2006) 'Design patterns from biology for distributed computing', *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 1, No. 1, pp.26–66.

Babaoglu, O., Meling, H. and Montresor, A. (2002) 'Anthill: a framework for the development of agent-based peer-to-peer systems', *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria: IEEE Computer Society, pp.15–22.

Bernon, C., Gleizes, M-P., Peyruqueou, S. and Picard, G. (2003) 'ADELFE: a methodology for adaptive multi-agent systems engineering', in P. Petta, R. Tolksdorf and F. Zambonelli (Eds.) *Engineering Societies in the Agents World III*, Vol. 2577 of *LNCS (LNAI)*, Springer, pp.156–169, *3rd International Workshop* (ESAW, 2002), Madrid, Spain, 16–17 September 2002, Revised papers.

Bernon, C., Gleizes, M-P. and Picard, G. (2007) 'Enhancing self-organising emergent systems design with simulation', in G.M. O'Hare, M.J. O'Grady, A. Ricci and O. Dikenelli (Eds.) *Engineering Societies in the Agents World VII*, Vol. 4457 of *LNCS (LNAI)*, Springer, pp.284–299, *7th International Workshop* (ESAW, 2006), Dublin, Ireland, 6–8 September 2006, Revised selected and invited papers.

Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999) 'Swarm intelligence: from natural to artificial systems', *Santa Fe Institute Studies in the Sciences of Complexity*, New York, NY: Oxford University Press.

Camazine, S., Deneubourg, J-L., Franks, N.R., Sneyd, J., Theraulaz, G. and Bonabeau, E. (2001) 'Self-organization in biological systems', *Princeton Studies in Complexity*, Princeton, NJ: Princeton University Press.

Casadei, M., Gardelli, L. and Viroli, M. (2007) 'Simulating emergent properties of coordination in Maude: the collective sort case', *Electronic Notes in Theoretical Computer Science*, Vol. 175, No. 2, pp.59–80, *5th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2006)*.

Cicirello, V. and Smith, S. (2004) 'Wasp-like agents for distributed factory coordination', *Autonomous Agents and Multi-agent Systems*, Vol. 8, No. 3, pp.237–266.

De Wolf, T. and Holvoet, T. (2007) 'Design patterns for decentralised coordination in self-organising emergent systems', *Engineering Self-organising Systems*, Vol. 4335 of *LNCS (LNAI)*, Springer, pp.28–49, *4th International Workshop on Engineering Self-organising Applications (ESOA'06)*, Hakodate, Japan, 9 May 2006.

De Wolf, T., Holvoet, T. and Samaey, G. (2006) 'Development of self-organising emergent applications with simulation-based numerical analysis', in S.A. Brueckner, G. Di Marzo Serugendo, D. Hales and F. Zambonelli (Eds.) *Engineering Self-organising Systems*, Vol. 3910 of *LNCS (LNAI)*, Springer, pp.138–152, *3rd International Workshop (ESOA 2005)*, Utrecht, the Netherlands, July 2005, Revised selected papers.

Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C. and Chrétien, L. (1991) 'The dynamics of collective sorting: robot-like ants and ant-like robots', in J-A. Meyer and S.W. Wilson (Eds.) *From Animals to Animals: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, Classics, Cambridge, MA: MIT Press, pp.356–363.

Fortino, G., Garro, A., Russo, W., Caico, R., Cossentino, M. and Termine, F. (2006) 'Simulation-driven development of multi-agent systems', in A. Genco, A. Gentile and S. Sorce (Eds.) *Industrial Simulation Conference 2006 (ISC 2006)*, The European Simulation Society (EUROSIS) and The European Technology Institute (ETI), Palermo, Italy, pp.17–24.

Gardelli, L., Viroli, M., Casadei, M. and Omicini, A. (2007a) 'Designing self-organising MAS environments: the collective sort case', in D. Weyns, H.V.D. Parunak and F. Michel (Eds.) *Environments for Multi-agent Systems III*, Vol. 4389 of *LNCS (LNAI)*, Springer, pp.254–271, *3rd International Workshop (E4MAS 2006)*, Hakodate, Japan, 8 May 2006, Selected revised and invited papers.

Gardelli, L., Viroli, M. and Omicini, A. (2007b) 'Design patterns for self-organising systems', in H-D. Burkhard, R. Verbrugge and L.Z. Varga (Eds.) *Multi-agent Systems and Applications V*, Vol. 4696 of *LNCS (LNAI)*, Springer, pp.123–132, *Proceedings, 5th International Central and Eastern European Conference on Multi-agent Systems (CEEMAS'07)*, Leipzig, Germany, 25–27 September.

Gardelli, L., Viroli, M. and Omicini, A. (2006) 'On the role of simulations in engineering self-organising MAS: the case of an intrusion detection system in TuCSoN', in S.A. Brueckner, G. Di Marzo Serugendo, D. Hales and F. Zambonelli (Eds.) *Engineering Self-organising Systems*, Vol. 3910 of *LNCS (LNAI)*, Springer, pp.153–168, *3rd International Workshop (ESOA 2005)*, Utrecht, the Netherlands, 26 July 2005, Revised selected papers.

Grassé, P-P. (1959) 'La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: Essai d'interpretation du comportement des termites constructeurs', *Insectes Sociaux*, Vol. 6, No. 1, pp.41–80.

Hinton, A., Kwiatkowska, M., Norman, G. and Parker, D. (2006) 'PRISM: a tool for automatic verification of probabilistic systems', in H. Hermanns and J. Palsberg (Eds.) *Tools and Algorithms for the Construction and Analysis of Systems*, Vol. 3920 of *LNCS*, Springer, pp.441–444, *Proceedings, 12th International Conference (TACAS 2006)*, held as part of the *Joint European Conferences on Theory and Practice of Software (ETAPS 2006)*, Vienna, Austria, 25 March–2 April.

Kephart, J.O. and Chess, D.M. (2003) 'The vision of autonomic computing', *Computer*, Vol. 36, No. 1, pp.41–50.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) 'Optimization by simulated annealing', *Science*, Vol. 220, No. 4598, pp.671–680.

Kulkarni, V.G. (1995) *Modeling and Analysis of Stochastic Systems*, London, UK: Chapman & Hall, Ltd.

Mamei, M. and Zambonelli, F. (2005) 'Programming stigmergic coordination with the TOTA middleware', *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, New York, NY: ACM Press, pp.415–422.

Menezes, R. and Tolksdorf, R. (2003) 'A new approach to scalable Linda-systems based on swarms', *ACM Symposium on Applied Computing (SAC'03)*, New York, NY: ACM Press, pp.375–379.

Molesini, A., Omicini, A., Denti, E. and Ricci, A. (2006) 'SODA: a roadmap to artefacts', in O. Dikenelli, M-P. Gleizes and A. Ricci (Eds.) *Engineering Societies in the Agents World VI*, Vol. 3963 of *LNCS (LNAI)*, Springer, pp.49–62, *6th International Workshop (ESAW 2005)*, Kusadasi, Aydın, Turkey, 26–28 October 2005, Revised, selected and invited papers.

Molesini, A., Omicini, A. and Viroli, M. (2008) 'Environment in agent-oriented software engineering methodologies', Special Issue on 'Engineering environments for multiagent systems', *International Journal on Multiagent and Grid Systems*, in press.

Mondada, F., Pettinaro, G.C., Guignard, A., Kwee, I.W., Floreano, D., Deneubourg, J-L., Nolfi, S., Gambardella, L.M. and Dorigo, M. (2004) 'SWARM-BOT: a new distributed robotic concept', *Autonomous Robots*, Vol. 17, Nos. 2–3, pp.193–221.

Omicini, A., Ricci, A. and Viroli, M. (2006) 'Agens Faber: toward a theory of artefacts for MAS', *Electronic Notes in Theoretical Computer Sciences*, Vol. 150, No. 3, pp.21–36, *Proceedings, 1st International Workshop 'Coordination and Organization' (CoOrg 2005)*, COORDINATION 2005, Namur, Belgium, 22 April 2005.

Omicini, A. and Zambonelli, F. (1999) 'Coordination for internet application development', Special Issue 'Coordination mechanisms for web agents', *Autonomous Agents and Multi-agent Systems*, Vol. 2, No. 3, pp.251–269.

Phillips, A. and Cardelli, L. (2004) 'A correct abstract machine for the stochastic Pi-calculus', in A. Ingolfsdottir and H.R. Nielson (Eds.) *Workshop on Concurrent Models in Molecular Biology (BioConcur 2004)*, CONCUR 2004, London, UK.

Ricci, A., Omicini, A. and Denti, E. (2002) 'Virtual enterprises and workflow management as agent coordination issues', Special Issue 'Cooperative information agents – best papers of CIA 2001', *International Journal of Cooperative Information Systems*, Vol. 11, Nos. 3–4, pp.355–379.

Ricci, A., Viroli, M. and Omicini, A. (2006) 'Programming MAS with artifacts', in R.P. Bordini, M. Dastani, J. Dix and A. El Fallah Seghrouchni (Eds.) *Programming Multi-agent Systems*, Vol. 3862 of *LNCS (LNAI)*, Springer, pp.206–221, *3rd International Workshop (PROMAS 2005), AAMAS 2005*, Utrecht, the Netherlands, 26 July 2005, Revised and invited papers.

Sauter, J.A., Matthews, R.S., Parunak, H.V.D. and Brueckner, S. (2005) *Performance of Digital Pheromones for Swarming Vehicle Control*, in F. Dignum, V. Dignum, S. Koenig, S. Kraus, M.P. Singh and M. Woolridge (Eds.) *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, the Netherlands: ACM Press, pp.903–910.

Solé, R.V. and Bascompte, J. (2006) 'Self-organization in complex ecosystems', *Monographs in Population Biology*, Princeton, NJ: Princeton University Press, Vol. 42.

Steward, S. and Appleby, S. (1994) 'Mobile software agents for control of distributed systems based on principles of social insect behaviour', *International Conference on Communications Systems (ICCS'94)*, Singapore: IEEE, Vol. 2, pp.549–553.

Uhrmacher, A.M. (2002) 'Simulation for agent-oriented software engineering', in W. Lunceford and E. Page (Eds.) *1st International Conference on Grand Challenges for Modeling and Simulation*, SCS, San Diego, San Antonio, Texas.

Viroli, M., Casadei, M. and Gardelli, L. (2007a) 'A self-organising solution to the collective sort problem in distributed tuple spaces', *2007 ACM Symposium on Applied Computing (SAC 2007), Special Track on Coordination Models and Languages*, ACM, Seoul, Korea, pp.354–359.

Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K. and Zambonelli, F. (2007b) 'Infrastructures for the environment of multiagent systems', Special Issue on 'Environments for multi-agent systems', *Autonomous Agents and Multi-agent Systems*, Vol. 14, No. 1, pp.49–60.

Weyns, D., Omicini, A. and Odell, J. (2007) 'Environment as a first-class abstraction in multi-agent systems', Special Issue on 'Environments for multi-agent systems', *Autonomous Agents and Multi-agent Systems*, Vol. 14, No. 1, pp.5–30.

Weyns, D., Schelfthout, K., Holvoet, T. and Lefever, T. (2005) 'Decentralized control of E'GV transportation systems', in F. Dignum, V. Dignum, S. Koenig, S. Kraus, M.P. Singh and M. Woolridge (Eds.) *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, the Netherlands: ACM Press, pp.67–74.

Zambonelli, F., Jennings, N.R. and Wooldridge, M.J. (2003) 'Developing multiagent systems: the Gaia methodology', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 12, No. 3, pp.317–370.