Case Studies for Self-Organization in Computer Science

Marco Mamei[a], Ronaldo Menezes[b], Robert Tolksdorf[c] and Franco Zambonelli[d]

[a]Dipartimento di Scienze e Metodi dell'Ingegneria, Universitá di Modena e Reggio Emilia
Via Allegri 13, 42100 Reggio Emilia, Italy, *marco.mamei@unimore.it*

[b]Florida Tech, Department of Computer Sciences
150 W. University Blvd., Melbourne, FL 32901, USA, *rmenezes@cs.fit.edu*

[c]Freie Universität Berlin, Institut für Informatik, AG Netzbasierte Informationssysteme,
Takustr. 9, D-14195 Berlin, Germany, *research@robert-tolksdorf.de*

[d]Dipartimento di Scienze e Metodi dell'Ingegneria, Universitá di Modena e Reggio Emilia
Via Allegri 13, 42100 Reggio Emilia, Italy, *franco.zambonelli@unimore.it*

1. Introduction

Self-organization can be defined as a process in which the internal level of organization of a *system* increases automatically without being guided or managed by an outside source [1]. Self-organization has been discussed, since long, in a number of disciplines ranging from physics to biology and recently, the popularity of the term has increased and it has even appeared in the popular vocabulary. We can currently observe a swarm — to paraphrase the term — of books being published to explain this, so called by the authors, fantastic phenomenon, see [2–7] for titles that focus in the swarming approach to self-organization. These are in addition to the few books in the field discussing the topic from a more academic perspective [8–12]. In the last few years, we have witnessed the birth of nature-inspired self-organization also as a field of computer science. The reason for this upspring is easy to state: the dynamism of IT scenarios like worldwide Internet computing and pervasive and ubiquitous computing makes it impossible to think in advance of the complex chain of interactions, and the cascading side effects arising in complex distributed applications. For this reason, the complexity of software applications can no longer be managed with traditional top-down techniques. Instead, components must be able to take care of themselves and handle unanticipated dynamic situations. The self-organizing phenomena found in nature are a perfect example of these capabilities. Insect colonies and biological cells are able to organize their activities and achieve tasks with exceptional robustness despite harsh and dynamic environments.

This paper aims to present a review on the state of the art of nature-inspired self-organizing mechanisms in computer science. We explore where such mechanisms may be a valuable – or at least noteworthy – alternative to classical approaches. The paper is divided as follows: in Section 2 we discuss on the fundamental role of interaction mechanisms with regard to self-organization. Moreover, we present a taxonomy to classify such interaction mechanisms that emphasizes their role in supporting self-organization. In Section 3 we review, with the help of the introduced taxonomy, a number of nature-inspired self-organizing mechanisms. In Section 4, we present some of the main research efforts trying to apply self-organization concepts to develop computer science applications. Finally, Section 5 contains our concluding remarks.

2. Interaction in Self-Organization

Natural self-organizing systems are constituted by a large number of individuals who interact

and coordinate to achieve tasks exceeding their capabilities as single individuals. These systems can scale to enormous sizes since interaction is local between individuals, rather than using some global medium. The way in which the individuals interact is of paramount importance and most of the benefits involved in self-organization are actually due to the right way of interacting. When we talk about interaction and coordination we refer to any kind of mechanisms allowing some individuals to orchestrate and influence each other's activities. In this section, we study the space of possible interaction mechanisms. We classify the interaction mechanisms according to a specific taxonomy to illustrate their strengths and weaknesses. This study and the resulting taxonomy will ground the discussion in the next section.

Among all the possible interaction mechanisms, we will focus only on uncoupled and anonymous ones. Uncoupled and anonymous interaction can be defined by the fact that the two interaction partners need neither to know each other in advance, nor to be connected at the same time in the network. For example, an interaction based on a sort of *message in a bottle* clearly belongs to this category. Uncoupled and anonymous interaction has many advantages. It naturally suits open systems, where the number and the identities of components are not known at design time. It suits also dynamic scenarios in that components can interact also in presence of network glitches an temporary failures. Moreover, it fosters robustness in that components are not bound to interact with prespecified partners, but interact with whoever is available. Summarizing, uncoupled and anonymous interaction is suited in those dynamic scenarios where an unspecified number of possibly varying components need to coordinate and self-organize their respective activities. Not surprisingly, this kind of interaction is ubiquitous in nature; cells, insects and animals adopt it to achieve a wide range of self-organizing behaviors The above cited *message in a bottle* is just an example of this kind of mechanisms and many more are conceivable. In order to proceed systematically, to differenciate mechanisms and to order our presentation it is useful to create a taxonomy. It is based on the answers to the following three questions: *(i)* what information is communicated, *(ii)* what is the flow of information, *(iii)* how is information used. We use it to order the possible uncoupled and anonymous interaction mechanisms and to highlight their strengths and weaknesses to support self-organizing systems.

2.1. What information is communicated?

Two kinds of communicated information can be distinguished [13]: *(i) Marker-based Interactions*: the information being communicated consists of special markers that agents use explicitly for the purpose of interaction. This is a kind of explicit interaction: actions like emitting sounds or pheromones are explicitly taken to communicate.

(ii) Sematectonic Interactions: the information communicated is the current state of the environment. This is a kind of implicit interaction: the local configuration of what is being done guides the actions. For example, when sorting items, the fact of placing similar items at a certain spot is not only a part of the solution, it is also a "signal" saying: "this kind of stuff goes there".

Sematectonic interaction is inherently *cheaper* than marker-based one. In fact, while in marker-based mechanisms, the interaction is an additional task that the agents have to undertake (i.e., they have to actively produce some marker), in sematectonic mechanisms, the interaction is embedded in the actual problem-solving task. On the other hand, marker-based interaction is more expressive than sematectonic one, in that the interaction-part can arbitrarily describe the problem-solving activities. While it is always possible to recreate a sematectonic interaction adopting a marker-based approach (e.g., markers can describe the current state of the solution), the opposite is not true (e.g., with markers is it possible to differentiate two solutions' states that look the same). Marker-based approaches are rooted on a shared communication facility where agents can post and retrieve information. Sematectonic interaction requires that the solution space is shared and observable by the agents.

2.2. What is the flow of information?

(i) Serendipitous: following this interaction mechanism, agents produce data and leave it in

the environment. This environment is conceptually a space in which agents and data etc. are situated. It might also be represented by multiple spaces, so that data and agents can be conceptually situated in several localities. Agents need to actively search for information in that space, which remains at its initial location. For example, as it will be detailed in the following, in the Linda approach [14], agents can leave tuples, containing information, in a shared data space, and can access tuples left by other agents through a pattern-matching mechanism. In this approach there is not an explicit way to direct a message to a specific recipient. The communication happens serendipitously.

(ii) Diffusion: here data is diffused across the environment to a large number of agents. This resembles broadcast "communication" where information is sent to whoever listens.

An interaction mechanism based on the diffusion of information requires a communication infrastructure more powerful than a serendipitous mechanism. In the diffusion case data needs to be transmitted to where the agents are. In the serendipitous approach, data can remain confined within a specific storage facility waiting for the agents to come by.

2.3. How information is used?

(i) Trigger-based: the reading of specific data triggers some actions on the agent. This is a kind of event notification and is the standard approach to agent coordination. For example, to coordinate movements, an agent can interact with another agent by leaving a message with something like "go to coordinate (100,100)". On the basis of the information received, the latter agent can proceed to the new target.

(ii) Follow-through: the interaction is not based on individual events triggering "one-shot" reactions. Instead the information being communicated drives the agent actions step-by-step. For example, to coordinate movements, an agent can leave a trail to be followed by another agent later on. In this way, the perception of the trail does not trigger single reactions, but constantly guides the agent activities.

All the above answers are not a complete and exhaustive list of all the possible alternatives. They are just landmark points and also intermediate and mixed approaches are conceivable.

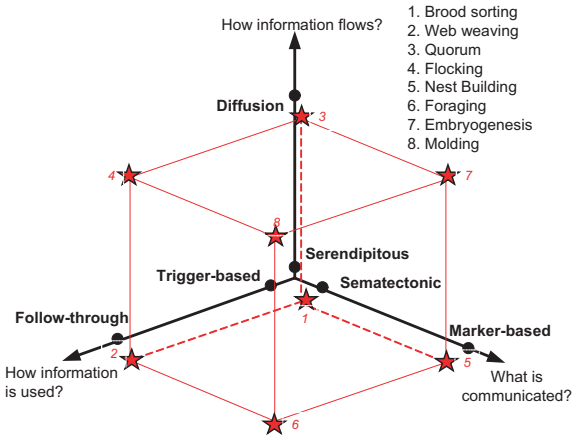

Figure 1. Interaction mechanism and associated self-organizing mechanisms

On the basis of the three questions, we can classify and represent different interaction mechanisms in a 3-dimensional space (see Figure 1). In the following section, we will show how most of the identified self-organizing strategies in nature fit well in this taxonomy.

3. Self-Organization Mechanisms

Self-organization is a phenomenon that takes place in several biological multi-agent systems. Social insects, by self-organizing their activities, achieve goals that overcome by far their capabilities as single individuals [9,15]. Other animals also demonstrate self-organized behavior. For instance large herds are completely decentralized and not influenced by any kind of leader, and still are able to perform complex coordinated activities. A lot of the self-organized phenomena found in nature provide a useful inspiration to solve computer science problems. In the following we will survey different self-organization mechanisms showing how they fit in the sketched survey-space.

3.1. Foraging

Foraging individuals self organize their activities to stochastically explore some environment in search of food or other resources [9,15]. An example are ants that leave a pheromone path in the environment after they found food found and are on the return to their nest. The existence of pheromones has an influence on the stochastic decision of other forager individuals — the more pheromone available the more likely an individual crossing the pheromone trail will choose that path. The pheromone is volatile and disappears from the environment after some time. This is negative feedback and results in old paths to food being forgotten if not exploited anymore. We call mechanisms similar to ant foraging by pheromones "foraging" in the following, knowing that there are foraging processes in biology that do not use pheromones.

Considering our taxonomy, foraging is *marker-based* in that the pheromones emitted by the individuals are explicitly conceived to enable interactions. It is *serendipitous* since an individual wanders randomly until it crosses by chance a pheromone trial. It is *follow-through* in that sensing a pheromone does not trigger one-shot reactions. Rather, pheromones form distributed structures like trails to be continuously followed.

The foraging mechanism finds natural applications in a wide number of computer science scenarios, ranging from network routing to optimization algorithms.

3.2. Nest Building

Another interesting way of self organization is the mechanism by which a swarm of individuals can cooperatively build a nest or another complex structure. The building can be realized without the presence of any cental engineer or master plan, but relying on an effective interaction mechanism and some simple local rules followed by the individuals. In a wide range of natural systems the building process is performed following to this schema: *(i)* individuals create small "bricks" (typically made of bodily waste), which contains pheromones. *(ii)* They wander randomly, but prefer the direction of the strongest local pheromone concentration. *(iii)* At each step, they decide stochastically whether to deposit the current brick. *(iv)* The probability of making a deposit increases with the local pheromone density.

This algorithm leads to the generation of scattered initial deposits. These deposits attract individuals and increase the probability that these individuals will leave a brick there. The most recent deposits are placed at the center of the pile. So, they are the strongest and piles tend to grow vertically rather than horizontally, forming columns. When columns grow near, the scent of each attracts individuals of the other, thus pulling subsequent deposits into the shape of an arch. For example, tropical termites can build nests more than 15 feet high and 10 tons in weight. These nests are multi-story structures providing storage for food, housing the brood, and protection for the population.

In out taxonomy, nest-building is rooted on a *marker-based, serendipitous, trigger-based* interaction mechanism. It is *marker-based* in that the pheromones left by individuals in bricks are explicitly conceived to enable interactions. It is *serendipitous* in that individuals wander randomly until they find pheromones. It is *trigger-based* in that pheromones do not form paths that constantly drive individual activities, but the presence of single pheromones triggers the release of the brick.

This kind of mechanism, as detailed in the next section, finds natural applications in robotics — i.e., robots collectively building some artifact.

3.3. Molding and Aggregation

Molding is when individuals are attracted by specialized food sources or other resources available in the environment [16]. If food is plentiful the individuals wander autonomously looking for food. When food becomes scarce, the individuals move toward one another forming a cluster. Then the cluster begins crawling the environment as a single super cell looking for a more favorable spot. When it finds such a place, it differentiates again, individuals are re-created and a new cycle begins. During the phase of the molding individuals produce a specialized chemical that diffuses in the neighborhood. Individuals are also attracted

by the same pheromone. They follow a gradient of the chemical, tending to aggregate where the chemical is the strongest. The molding process is exploited by several bacteria and microorganisms to survive harsh environments.

Molding with the additional connotation of aggregation is rooted on a *marker-based* interaction mechanism in that is uses a pheromone conceived for that purpose. It is based on *diffusion*in that the pheromone diffuses in the environment creating a gradient. It is *follow-through* since individuals are continuously driven by the gradient.

This kind of mechanism finds natural applications in robotics and team coordination — for example, a rescue team can decide to aggregate when particularly difficult conditions are met.

3.4. Morphogenesis

Morphogenesis is one of the major outstanding problems in the biological sciences. It concerns the fundamental question of how biological form is generated starting from an immense number of biological cells that self-organize their activities. For example a wide range of species use morphogen gradients to determine positional information of cells. Cells at one end of the embryo emit a morphogen that diffuses along the length of the embryo. The concentration of this morphogen is used by other undifferentiated cells to determine their distance from the emitting end, and thus determining whether they lie in the head, thorax or abdominal regions [17]. Different morphogens are used for determining the position along other dimensions. This mechanism has been widely studied — also experimentally — in the *Drosophilae* embryo.

In our taxonomy, this mechanism is *marker-based* in that it involves specialized chemicals (the morphogen gradients) to communicate. It is based on *diffusion* in that morphogen chemicals diffuse to enable long range communications. It is *trigger-based* in that the chemical induce one shot cell differentiation.

This mechanism can be fruitfully applied in a number of computer science scenarios such as in modular robotics and self-assembly. In these applications a swarm of robots uses virtual morphogens to achieve a final global shape.

3.5. Web Weaving

Web weaving is the mechanism in which individuals build collectively webs that are used both to easily roam across an environment and to capture preys passing nearby. The web weaving activity is based on the low-level task of connecting two ground spots with a dragline. The web, in the end, is composed by an aerial network of draglines, hooked to fixed spots on the ground. It is important to remark that, once fixed, a dragline provides a new, shortest path between two spots. Similarly to the foraging mechanism, individuals roam randomly across the environment weaving a dragline between two random spots. However, individuals are attracted by already existing draglines and prefer walking upon one of them rather than on the ground. This facilitates the chance of having a dragline beginning where another ends, and thus facilitates the web creation. In biological systems, the likelihood of walking upon a dragline has been carefully tuned. If it is too low, no web is built and all available space filled with disconnected segments. If it is too high, individuals are trapped in their own draglines and no real weaving occurs. This mechanisms is exploited by several spider species around the world. The *Anelosium eximius* which can be found in French Guiana is one of them. The individuals live together, share the same web and cooperate in various activities.

Following our taxonomy, there is an interaction mechanism that is *sematectonic*, in that the very mechanism used to communicate is the web that individuals are building. It is *serendipitous*, in that individuals find each other silk draglines by chance. It is *follow-through* in that the web does not trigger one-shot reactions, but guides constantly individuals movements and weaving.

This mechanism can be applied to file retrieval in peer-to-peer systems and network routing. Virtual draglines can connect host or resources to allow them to be quickly discovered and retrieved.

3.6. Brood Sorting

Brood sorting is the mechanism in which individuals self-organize their activities to collectively sort items such as eggs and microlarvea. Enabling this operation is that individuals just wander ran-

domly and pick up and drop items according to the number of similar surrounding objects. For example, if an individual finds a large cluster of similar items together with a different one, it will most likely pick up the misplaced item and start roaming around. That individual will probably deposit its load in a region containing other items similar to the one he is carrying. The workers of the ant *Leptothorax unifasciatus* sort the colonys brood adopting this mechanism.

In our taxonomy this is *sematectonic* since the sorting process itself drives coordination. It is *serendipitous* because individuals have to stumble onto a pile of e.g. larvae to coordinate. It is *trigger-based* in that a pile of items triggers one-shot pick-up or release of the items. This mechanism can represent a solution to database organization and virus protection. A swarm of computational agents can review system resources, cluster them and eventually detect outliers.

3.7. Flocking, Schooling and Herding

Flocking is a self-organized behavior that takes place when individuals are driven by three local forces: collision avoidance, speed matching and flock centering. Schooling and herding are driven by similar forces [18]. Collision avoidance consists of individuals being programmed to pull away before crashing into one another. Speed matching relates to the attempt of the individuals to go at the same speed of their neighbors. Centering is the idea that individuals have a notion of the center of the flock and that they always try to move toward the center. More importantly, this coordinated movement happens without a leader — any individual can maneuver at any time [10]. More recently, Toner and Tu [19] have demonstrated that flocks never attain equilibrium, thus the flock is an entity that is continuing in characteristic but unpredictable in pattern formation. Flocking as well as schooling are common to simulate birds and fish behavior. As opposed to foraging, individuals here move less randomly. The attractiveness of flocks and schools to individuals is directly but not linearly proportional to its size [20]. In these systems, the presence of sentinels is also common. Sentinels in a herd are individuals that under certain environmental conditions are more sensitive to stimuli like sound, smell, approach of danger, and react earlier than other individuals [21]. After the sentinel moves, others tend to follow due to the forces mentioned above.

Following our taxonomy, flocking is *sematectonic* in that is the very flocking formation that drives coordination. It is *based on diffusion* in that information (visual information or wind turbulence) spread in space and can be sensed by other individuals at a distance. It is *follow-through* because it continuously drives motion.

This mechanism can represent a solution to motion coordination in a wide range of scenarios. For example a swarm of rovers or UAVs (Unmanned Airspace Vehicles) could use a flocking mechanism to roam across an environment.

3.8. Quorum

Quorum is the mechanism in which individuals self-organize their activities to sense how many other similar individuals are around them and react accordingly. The reaction normally relates to performing some action once a specific *quorum* is reached. This mechanism is possible because the actions of individuals diffuse some signal in their neighborhood and the density of the signal triggers the action. As in foraging, the signal is assumed to disappear over time, however quorum normally triggers an action that has a fixed duration in time, meaning that the speed at which the signal fades is not as crucial as in foraging. Quorum is common in bacterial individuals such as the *vibrio fisheri*, a species of bioluminescent bacteria found in several niches in the marine environment; they reside in the light-emitting organs of certain species of squids and fish of the deep ocean. They use quorum to decide when to produce bioluminescence; they release a bioluminescent hormone while monitoring the levels bioluminescence in the environment.

Following our taxonomy, Quorum is rooted on an interaction mechanism that is *sematectonic* in that the action used to interact is also the one that individuals want to coordinate (e.g. light-emissions in *vibrio fisheri*). It is *based on diffusion* (e.g. in the *vibrio fisheri* the bioluminescent hormone diffuses across the population). It is *trigger based*; receiving light information triggers actions.

4. Current Research

Several self-organization mechanisms appear very suitable in handling a number of relevant computer science scenarios. In the following we review applications for self-organization in several areas of computer science by referring to concrete existing work or illustrate it by possible applications that can be easily conceived. Evidence on the specific implementations of these scenarios is subject to future work.

Five main areas could naturally benefit from the presented self organized behaviors: *middleware*, *information systems and management*, *security*, *robotics*, and *network management*. Each area will be discussed in the next subsections.

4.1. Middleware

The development of middleware is essential to almost to all fields of applied computer science, but they are particularly important to distributed systems and multi-agent systems as they encapsulate mechanism to allow the coordination and organization of processes and data respectively. Self-organization in middleware is becoming more attractive since the complexity of the problems in the field goes beyond the capabilities of standard approaches. In this section we divide the area of middleware into four application areas namely Grid Computing, Coordination Systems, Cache Replacement Systems, and Pervasive Computing.

4.1.1. Grid Computing

The general ideas of Grid Computing can be traced as far back as the seventies with researchers at Xerox PARC who envisioned a "worm" that could travel in a distributed system to utilize the computers idle time. Project like Condor [22] were proposed in the eighties. Grid Computing has been more of a "promise" in computer science than "reality". Despite the many success stories [23–25], the original idea of integrating heterogeneous systems in this nearly seamless framework has not happened at the pace it was expected. For such a framework, computing resources need to become a commodity in which users do not need to worry about *how* it works but only about *what* they need and *how much* they can afford to pay for what they need.

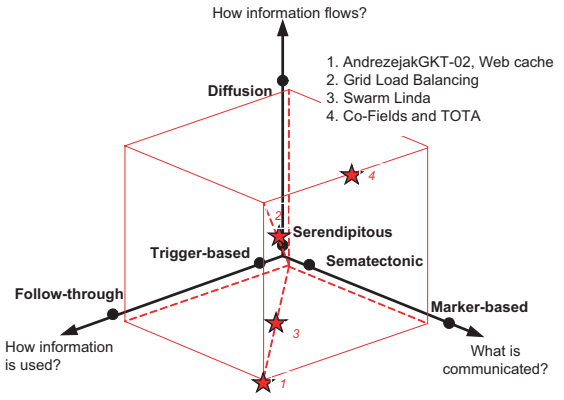

Figure 2. Middleware infrastructures in the taxonomy space.

Resource utilization is the *sine qua non* to performance in Grid environments: computers need to be well utilized, data need to be located where it is mostly needed, and services need to be easily found. At the same time this has to occur in a very dynamic environment, since computers in Grids can even be desktops with up-times that are not guaranteed The resource utilization need to be adaptive to cope with such conditions while maintaining system with small overhead.

Self-organized mechanisms such as foraging, molding, and brood sorting offers a way to rethink resource utilization in Grid frameworks. Foraging is a desirable mechanism to emerge solutions for finding and placing resources. A well implemented foraging approach can yield solutions that enable programs to locate resources with near minimum effort. One can envision a grid framework in which histories of where services have been located in the past are used as a positive reinforcement to the decision of where to look for such services. Such histories can have a pheromone-like characteristics forcing the system to "forget" old histories. For instance, [26] looked at the use of foraging as a distribution mechanism for services (see Figure 2).

Molding and brood sorting are more adequate

to devise an approach to organize grid resources according to similarities. Brood sorting can be the basis for organizing data in certain locations in the Grid according to similarities or based on a need by processes providing a emergent solution to the problem of load-balancing. Molding on the other hand can be used in conjunction to brood-sorting as a controlling mechanism to avoid over clustering. The differentiation mechanism in molding can serve the purpose of controlling the aggregation of data in only one location on the grid. As the cluster gets larger (thus moving towards becoming a bottleneck), molding can influence a new differentiation phase to form clusters in other locations in the grid. The emergent pattern is the formation of various clusters distributed on the Grid.

4.1.2. Coordination Systems

Tuple-based coordination models like Linda, have been successful in tackling the intricacies of medium-to-large-scale open systems [14]. The atomic units of interaction here are tuples which is a structured set of typed data items. Coordination activities between application agents (there included synchronization) can take place via indirect exchange of tuples through a shared tuple space. The coordination primitives access and manipulate a shared tuple space. A tuple can be written in the tuple space or retrieved via an associative — pattern matching — mechanism. Agents can coordinate their actions — in a completely uncoupled way — on the basis of the presence or the lack of specific tuples. Linda's tuple-space model is incorporated in commercial middleware platforms such as Jini [27] and GigaSpaces [28]. However, it has not gained widespread acceptance as middleware for large-scale distributed systems. Currently, the dominating options for implementing distributed tuplespaces are:

(i) Centralization is a simple client-server distribution strategy where one specific server-machine operates the complete tuple space. *(ii) Partitioning* of tuple spaces co-locates tuples with common characteristics in one of a a set of tuple-space servers. *(iii) Full replication* places complete copies of tuple spaces on several different machines. *(iv) Intermediate replication* has been proposed in the early of Linda [29]. The schema bases on a grid of nodes formed by logical intersecting "busses". Each node is part of exactly one *outbus* and one *inbus*. Data stored is replicated on all nodes of the outbus, whereas searches are performed on the inbus. As one inbus intersects all outbusses, it provides a complete view of the tuple space.

None of these approaches scale well with the number of processes in the system. SwarmLinda [30–33] is an attempt to transfer the *foraging* and *brood sorting* mechanisms from natural agent systems to the implementation of large-scale distribution of a tuplespace.

In SwarmLinda the partitioning of the tuple space is dynamic and based on the concept of brood sorting used by ants. The individuals that operate here are tuple-ants. The environment is the network of SwarmLinda nodes. The state is the set of tuples stored thus far. A SwarmLinda implementation may use brood sorting in the process of tuple distribution to group tuples based on their template which will lead to the formation of clusters of tuples.

The approach is able to improve the availability of the system without having to count on costly techniques such a replication of data. In the ant-based approach, there are no assumptions about the behavior of applications, there is no pre-defined distribution schema, and there are no special scenarios implemented to deal with failures in a node.

Ants look for food in the proximity of the anthill. Once found, the food is brought bach and a trail is left so that other ants may know where food can be found. They know the way back because they have a short memory of the last few steps they took and also because the anthill has a distinctive scent that can be tracked by the ants. In a tuple space context, foraging of ants can be abstracted by looking at tuples as food. The locations where the tuples are stored is the terrain while the templates are seen as ants that wander in the locations in search of tuples. The anthill is the process that executed the operation. The result of this is the emergence of application specific paths between tuple producers and con-

sumers. Since scents are volatile, the paths can dynamically adapt to changes in the system.

There are further occasions in the implementation of a Linda-like system to use self-organization. The named references give more details on how to show an adaptive behavior for collections of similar tuples by using tuple-ants and how to balance tuple- and template movement dynamically by using the ant-abstraction both for tuples and templates (see Figure 2).

4.1.3. Adaptive Web Cache Replacement

Part of the overall Web middleware are Web caches that minimize latency for Web requests. User-side caches act as proxies for servers and receive requests for information. They store a copy of it when transferring it to the client. When another client asks for the same information, it can be taken from the cache's store thereby minimizing the load for servers and the response time. Server-side caches are used to minimize load on servers for dynamic pages by storing the result for a retrieval for further accesses. They shield the server from the accesses that trigger processing, thereby minimizing the load of the servers and in turn lowering its response time for other dynamic information items. The cache tries to optimize the *hit rate* which measures the ratio of information items found in the cache versus those those that had to be retrieved from the actual target server. An important decision has to be taken when the limited cache size is taken up by stored copies of information entities and some have to be selected for removal. The choice is critical for the caches performance if the "wrong" objects are removed, namely those that will be retrieved again soon. There are numerous techniques for cache placement and replacement [34]. They all exhibit optimal performance only for specific characteristics of the stream of requests. Today, a Web cache implementations select one of those strategies statically, mostly the LRU strategy. It would be interesting to have the cache dynamically select a strategy for the current stream of requests.

The foraging mechanism appears suitable in this context. One can imagine that cache hits can be used as food for virtual ants representing web cache entries. As long as a virtual ant get enough food (i.e., hits) it remains in cache. The ants that die out of food are replaced. To enforce adaptiveness, we can let ants spread pheromones indicating where are the resources most likely to be accessed. Newly created ants could follow such pheromones to predict what resources will be accessed in the future, and to crate novel relevant entries in the cache [35] (see Figure 2).

4.1.4. Pervasive Computing Systems

Pervasive computing introduces some peculiar challenging requirements in the development of distributed software systems: *(i)* since new agents can leave and arrive at any time, and can roam across different environments, applications have to be adaptive, and capable of dealing with such changes in a flexible and unsupervised way; *(ii)* the activities of the software systems are often contextual, i.e., strictly related to the environment in which the systems execute (e.g., a room or a street), whose characteristics are typically a priori unknown, thus requiring to dynamically enforce context-awareness; *(iii)* the adherence to the above requirements must not clash with the need for promoting a simple programming model possibly yielding light infrastructures. Unfortunately, current practice in distributed software development, as supported by currently available middleware infrastructures, is unlikely to effectively address the above requirement: *(i)* application agents are typically strictly coupled in their interactions, thus making it difficult to promote and support spontaneous interoperations; *(ii)* agents are provided with either no contextual information at all or with only low-expressive information (e.g., raw local data or simple events), which are difficult to be exploited for complex coordination activities; *(iii)* due to the above, the results is usually an increase of both application and supporting environment complexity.

Field-based coordination relies on virtual computational fields (e.g., distributed data structures), mimicking gravitational and electromagnetic fields, as the basic mechanisms with which to coordinate activities in open and dynamic ensembles of application components. This enables components to spontaneously interact with each other via the mediation of fields and — as in phys-

ical systems — to self-organize their activity patterns. All of this with the additional advantage that — unlike in real-world physical systems — one can shape fields according to any needed virtual physical law, to achieve a variety of coordination patterns in support of a variety of application goals. Accordingly to the description given in Section 2, we can say that field based coordination takes inspiration from both the *molding* and the *morphogenesis* metaphors. The main value of fields relies in that they can create distributed data structures providing agents with the right contextual information at the right location. Such a contextual information can profitably guides the agents' activities and supports their coordination. The model Co-Fields [36] and the middleware TOTA [37] support the spreading of fields across an ad-hoc network of pervasive computing devices. TOTA tuples are characterized in terms of three parameters (C,P,M). C is the content of the tuple. It represents the information being carried on; such as the field properties and their magnitude. P is the propagation rule. It specifies how the tuple has to diffuse across the network. M is the maintenance rule, it specifies how the tuple should change upon network reconfigurations and external events. An API allows the creation of tuples — fields — and the injection of them in the network. Agents coordinate on the basis of the configuration of tuples — fields — sensed in their local environment. In a motion coordination application, for example, an agent could inject a tuple with an integer value that increases as the tuple propagates across the network. Another agent could follow the gradient of this tuple to discover and reach the location where the first agent is located (see Figure 2).

4.2. Information systems and management

4.2.1. Database Organization

Despite all research in database management systems, the main concepts are still based on IBM's original concept proposal — the first databases built on SQL in the market were Oracle version 2 and SQL/DS from IBM and they could handle very small quantities of data.

Nowadays, we are dealing with amounts of data that are hard to imagine and systems configurations that were never thought of by the first proponents of database ideas, like the the CERN project on the exabyte (10^{18} bytes) database for Large Hadron Collider (LHC) experiments [38]. The job of physically organizing large amounts of data is quite a challenge to be resolved because no single machine is be able to store them — hence distribution is *sine qua non*.

A similar problem is the organization of small quantities of data scattered over small devices. In these scenarios, devices (such as sensors can hold small amounts of data that need to be organized in a way so that queries can performed efficiently.

Self-organized approaches are excellent for data organization. Distributed database systems (DDS) is a collection of logically interrelated databases distributed across various physical locations [39]. In addition to the database itself, we need a distributed database management system. The data distribution aims at improving performance while maintaining transparency, reliability and cost savings. Self-organization inspired by *brood sorting* may be used to provide an adaptive distribution of data based on criteria based on the database tables, records, and even fields. The criteria can also be physical (e.g. secure data should be maintained in pre-defined secure sites)

The resulting distribution is adaptive to the activities being carried out, the size of the database, and other factors. One can envision an algorithm for data distribution based on brood sorting where each young corresponds to an element of the database. These elements have incentives to find locations where similar elements are located. To avoid bottlenecks we need to use scales of feedback where positive feedback becomes negative after some threshold. These concepts are very similar to the ideas of self-tuning databases such as the project COMFORT [40] which advocates that DBMSs should be without Data Base Administrators. Issues such as load balancing can be self-regulatory so that the performance of the system is kept at good levels despite the its usage. Requirements such as these are well addressed by self-organized approaches as the ones described above in which elements of the database are able to move in a reactive way depending on the load of the system (see Figure 3).

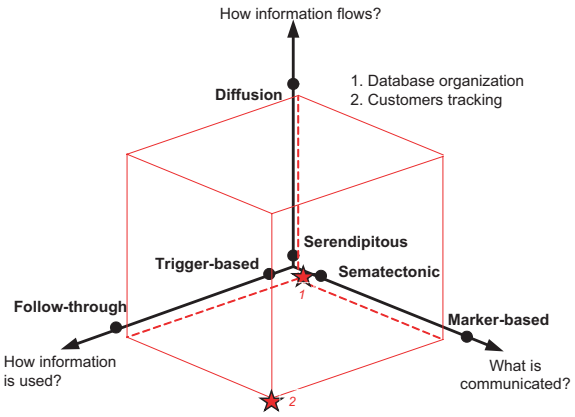

Figure 3. Information systems in the taxonomy.

4.2.2. Retail Stores

Optimizing the process of manufacturing and retail is of high economic value. The majority of the big shops use some mechanism to decide where the products are placed but few do this on-the-fly. The standard way of studying store efficiency, for instance, is to hire experts in the field and ask them to analyze the performance.

Supermarkets started to experiment with new technologies like RFID to help them control stock, know more about store efficiency, and aiding on decision making. In the UK, Marks and Spencer and Tesco have pioneered the use of RFID for stock control. A more impressive attempt is the Metro Future Store, located in Rheinberg, Germany. With the help of RFID technology, the store is able to provide stock inventory, shelf filling information, implement a complex costumer information system, and payment system.

It is surprising that not much has been published on how to address the issue of store organization based on costumer behavior. Self-organization may play an important role because people (costumers) in crowds behave very similar swarms of other animals. With RFID technology it is possible to detect the path for individual costumers and decide on product placement accordingly. Again here this would be similar to foraging where paths traversed by the costumer are being reinforced.

A specific model may also be used to analyze the effect of changes before implementing them because the model can predict the outcome. Due to the self-organizing nature of people, what may be good for one store may not be necessarily good for others. A system based on foraging will always converge to the best to the current scenario.

The possibilities are almost infinite: one can do profiling of costumers — what they buy, when they buy; the store can detect corelation between products which are used to have special offers displayed on the path of the costumer for products that he is likely to buy. The store may also be able to save on energy cost with an intelligence light or cooling/heating system.

4.3. Security

4.3.1. Malicious Code Protection

Even the most well developed critical system will contain some vulnerability. This fact is what drives malicious code writers to develop programs that can effectively exploit these vulnerabilities. To better illustrate the main problems of the current state of the art, we will briefly make a comparison with biological viruses and immune systems. A virus is a non-cellular biological entity that can only reproduce when inside a host cell. The host cell then bursts and release copies of the virus that will then attach to new cells, enter the cells and continue the cycle of bursting, release of copies, and so on. Our immune system is able to cope with viruses by recognizing infected cells (called antigens) and destroying them. Our immune system is able to cope with unknown viruses using a process of refinement of current phagocytes. The systems above only works because we do have an *adaptive* immune system. This is the opposite of what happens in a house fly for instance which is born with a pre-set number of defenses — a non-adaptive immune systems. Flies can evolve new protection mechanisms but only thought evolution of their generations. The scenario we find ourselves in computer science with regards to malicious code protection is similar to the one of a house fly. Our protection (anti-virus) are just like the immune systems of a fly

while the malicious code (computer viruses) are like the malices. Because anti-viruses contain a pre-defined set of protection rules, malicious code writers can exploit this non-adaptive characteristic of the software and write effective malicious code. Again, the protection system will only handle the new malice in new generations of the software — new anti-virus definitions.

Self-organization can be effectively used to provide adaptive protection against malicious code in the a similar way humans immune system can provide protection against antigens. The protection may works by distributing specialized agents in the network. These agents would recognize only few kinds of malicious code. The protection system enable agents to roam in search of known threats. Once an agent finds a malicious code it recognizes, it removes it from the system. This process is very close to *morphogenesis* because roaming agents may be seen as not-quite-specialized cells that only commit when a threat is identified. As in our immune system, these agents are attracted to the threat and will flow to areas of the network where the security has been breached. As a malicious code spreads, agents are able to replicate themselves to provide effective isolation of the compromised part of the system. Protection agents require the existence malicious code to replicate — low levels of a particular threat will cause the number of protection agents to be reduced. Agents could replicate if they sense large amounts of malicious code. This can be implemented by letting the malicious code act as a sort of pheromone. As its concentration increases it may trigger the replication of the the protection agents. New agents can also be introduced in the system at any point to add new protections. New versions of old agents can also be introduced and programmed to remove the old agents. Most important, systems need to react quickly to new threats. Each node of a network can contain a software stub which is in charge of letting the rest of the network know of unrecognized activities. This may be efficiently implemented by letting the stub diffuse the information on the network. Protection agents, being attracted by this diffused information, will flow to that location following the gradient of diffusion (similar to what is described for motion coordination and inspired in *molding* and *flocking*). The information collected by the software stubs may be used to aid industry in writing new protections.

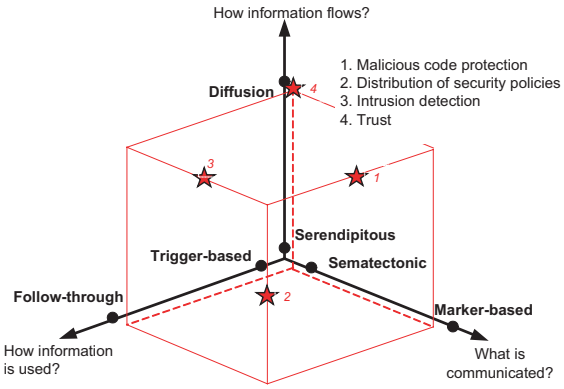

Figure 4. Security mechanisms in the taxonomy.

In the field of artificial immune systems [41], IBM [42] followed the idea of having a mechanism to distinguish in a systems what is self (and safe) from what is non-self (and consequently unsafe) (see Figure 4).

4.3.2. Distribution of Security Policies

Network security policies are one aspect of information which influences the behavior of objects within the system [43]. Human managers are adept at interpreting both formal and informal policy specifications and, if necessary, resolving conflicts when making decisions. However, the size and complexity of large distributed systems has resulted in a trend towards automating many aspects of management into distributed components. If policies are coded into these components they become inflexible and their behavior can only be altered by recoding [43]. Little has been done on the effectiveness of the update of the policies, and how this update can be carried out in an autonomous fashion.

Menezes [44] has described a self-organized solution for policy distribution based on *foraging*

with elements of *molding*. Let us first assume a network in which all the nodes currently have a copy of the security policy Φ — one can assume that a node that does not have any policy is similar to one that has an outdated policy. Let us also assign one of the nodes as the initiator of the policy update process into a Φ_n where a version n is more current than all versions $< n$. The distribution algorithm works by making the security policies active and attracted by older versions of policies. When a policy Φ_k is created, it looks at the neighbors and replicates itself to the neighbors that contain policies $\Phi_{<k}$. If the node finds nodes with policies $\Phi_{\geq k}$, the individual stops its activity since other individuals are already spreading on the network.

Molding is only utilized when there are failures on the network. Under normal circumstances, a foraging approach is used in the transition rule taking in consideration the policy version and the learned path from previous distribution. Our transition is given by:

$$p_{id}(t) = \frac{[\tau_{id}(t)]^{\alpha} \cdot [\delta_{id}(t)]^{\beta} \cdot [\eta_{id}(t)]^{\sigma}}{\sum_{h \in J_i} [\tau_{ih}(t)]^{\alpha} \cdot [\delta_{ih}(t)]^{\beta} \cdot [\eta_{ih}(t)]^{\sigma}}$$

In the equation above τ_{id} represents the learned behavior of the system. Each time a policy at node i chooses the neighbor $d \in J_i$ it reinforces the path $i \rightarrow d$ so as to indicate to other policies that the neighbor d was chosen at some point in the past. This controls the desirability of a node based on previous experiences of policies at that location. The term δ_{id} is responsible for the decision being made by the policy with regards to the version. The older the version at a given neighbor d, the more attractive d is. Last, the term η_{id} represents some static information about the network such as link reliability, or maximum available bandwidth which uses a local update technique each time a neighbor is chosen. This local update may modify the values of τ only. The value of δ is updated at defined intervals in which nodes "ping" their neighbors for that information.

4.4. Robotic Systems

The use of self-organization in robotics is current topic of intensive research. Several surveys exist covering this application of self-organization (e.g. [45], and a few sections in [9]), here we try to complement what has been already described with a more up-to-date view of the field. In general, the power of self-organization with regard to robotics is about flexibility and robustness. A group of autonomous robots self-organizing their activities are able to adapt to dynamic situations, faults and glitches. For example, if a robot breaks down, the others can self-reorganize their activities and take over its duties. Bonabeau *et al.*[9] emphasizes that there are several advantages in having self-organization implemented in robotic activities. In particular they discuss that randomness and fluctuations in individuals activities is an asset to problem solving, in that individuals become more flexible to changes, more reliable and more fault-tolerant. In the following, we will survey self-organizing approaches to tackle two exemplary application scenarios in multi-robot systems: motion-coordination and self-assembly.

4.4.1. Motion Coordination

The idea of motion coordination in robotics is to have multiple robots and any kind of unmanned vehicles (aerial and ground, UAV and UGV respectively) — coordinating their movements to achieve cooperative tasks like exploring an environment or patrolling an area.

A widely-exploited approach is based on the idea of having robots self-organize their movements on the basis of artificial potential fields that are digital analogues of the gravitational and electromagnetic fields [46]. Consider the case of a group of robots distributed in an environment, having to orient themselves and to coordinate their movements. To interact meaningfully, they use their sensing capabilities (e.g., cameras) to acquire contextual information. If they do not have direct means to communicate they necessarily have to coordinate with each other by detecting each other movements and adapt their behavior accordingly. Robots could take advantage from interpreting the environment and the other robots' behavior in terms of a field-based representation and acting on the basis of fields. An agent seeing an obstacle through its camera could represent it as a repelling field, whose strength in-

creases as the perceived dimensions of the obstacle increase. This approach promotes a clean separation of concerns between the phase of processing sensorial stimuli and the phase of reacting to it. Realizing motion coordination patterns with this approach is relatively easy. For example, if each robot emits an attracting field, the robots group at a single point. If each robot emits a repelling field, the robots tend to disperse in the environment — thus they explore the environment efficiently without overlapping.

Similar approaches to control robot movements have been proposed in [47,48]. There, a swarm of wirelessly interacting robots need to coordinate their movements to efficiently patrol an unknown environment. They use their wireless network to actually spread distributed data structures representing the field abstraction. Each robot stores a piece of such a distributed data structure and moves on basis of the pieces contained in the robots in its neighborhood. These approaches are clearly inspired by the molding and flocking mechanisms. Fields in fact are based on a marker-based interaction mechanisms. However, since their actual deployment depends on both the data structure and the current network topology (robots' positions), this approach has also sematectonic elements (see Figure 5).

[49] presents a foraging inspired approach to coordinate UAVs. It has been implemented by spreading pheromones in a virtual data-space shared among the UAV agents. A server holds a virtual copy of the environment the UAVs have to explore. UAVs connecting to that server can store pheromones at the location corresponding to their physical position. Moreover, they can access the resulting pheromone trails and move according to specific rules.

4.4.2. Self Assembly

In robotic self-assembly a large number of simple interacting micro-robots arrange their respective positions and connect in a global structure that is suitable for a specific task. For example, in a pipe repairing task, a swarm of micro-robots could be able to navigate in a pipeline, recognize the presence of holes, and self-assemble with each other so as to perfectly repair the pipe.

In [50] a method is presented to organize the growth of a 2D structure in a swarm of mobile robots. Robots are autonomous, can only sense their local environment, and are largely interchangeable. Each robot is defined by a state value and a lookup table of transition rules, which specify conditions under which a robot will connect itself to one of its neighbor robots. Connections, as prescribed by the lookup table, take place on the basis of a robot's internal state. Robots move randomly in a 2D square lattice around an initial robot, called seed, and when the conditions of a transition rule are met they attach themselves to neighbor robots. Similar, [51] reports on robots that build a 3D structure using building blocks of different types. Robots move randomly in a 3D lattice around an initial building block. They can sense the presence of building blocks or other robots within the local 3x3x3 lattice. When a robot finds a building block, it can pick it up and carry it toward the growing structure. Then, if the robot perceives a suitable environmental configuration it deposits the block, making the structure grow. These two approaches implement the Nest Building mechanism (see Figure 5).

In [52], each robot runs a simple finite state automaton in which state transitions are driven by the local state, the state of neighbor robots, their locations, and some external information. Communications are limited to the immediate neighborhood and a limited number of bits are exchanged at each time step. The goal is not to create an exact predefined shape, but a structure with structural or morphological properties. Any stable "emergent" structure with the desired properties is satisfactory, with no regard for the "optimality" or details of the resulting geometry. This approach is very similar to the nest building and web weaving mechanism.

The approach in [53] implements the molding mechanism and enables robots to create regular spatial distributions like hexagonal and square lattices. Each robot is like a particle with a mass and it is subject to "artificial physical" forces enabling robots, detecting only nearby robots' distances, to spread in the space, creating regular lattices. [54] and [55] realize molding by propagating multiple signals in the robots' ensemble.

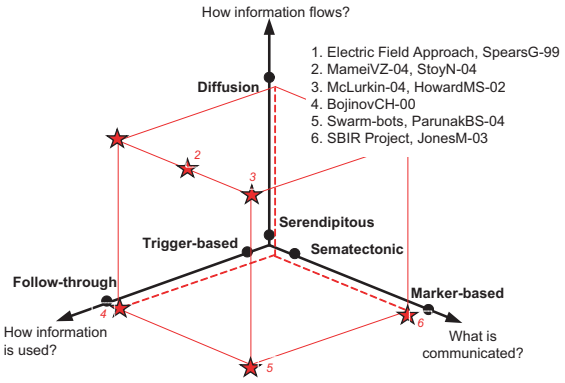

Figure 5. Robot mechanisms in the taxonomy.

Robots react to the current signal configuration depending on their internal state, realizing complex and articulated structures.

The Swarm-bots project [56,57] concentrates on the use of self-organization in the assembling of structures made of insect-sized active components. A Swarm-bot is a group of around 30 so called *s-bots* that self-assemble. The project focuses on the novel concept of *functional* self-assembly. Robots have to assemble in such a way it is useful for the task they are trying to achieve. For example, they can form a long chain to overpass a crack in the terrain or to climb a high step.

4.5. Networks

4.5.1. Mobile Ad-hoc Networks

Wireless ad hoc and sensor networks challenge current approaches to routing and network control Lacking any centralized point of control, nodes in ad hoc networks must cooperatively self-organize routing and medium access functions. Also, nodes in ad hoc networks may be mobile or ephemeral (like the limited lifetime of battery-powered sensors), inducing continual changes in the network topology.

Ad hoc network routing protocols usually dynamically build a sort of routing overlay structure for routing. "Ant Routing" [9] and "Gradient Routing" [58] make self-organization metaphors in building the overlay particularly explicit.

Ant routing algorithms apply foraging. Ant packets explore the best route between specified source and destination nodes. Instead of having a fixed next-hop destination, the routing table on nodes has multiple next-hop choices for a destination, with a value indicating the goodness of choosing a hop as the next one to form the route. These values are initially equal and updated according to the ant packets pass by. Given a specified source node and destination node, the source node will send out some kind of ant packets based on the values on its own routing table: the higher the value of a link, the more likely the ant packet will be sent to that link. Ants will explore the routes in the network recording the hops they have passed. When an ant packet reaches the destination node, the ant packet will return to the source node along the same route backward. The ant packet will change the routing table for every node increasing the value of the hop it comes from while decreasing the values of other links. Changing routing table is like laying down some virtual pheromone to affect the route of the subsequent ant packets. Since the route with higher value is favored, more ants will pick up that route, and the route will be strengthened. With this positive feed-back loop, we can expect a best path will be quickly found. Moreover, when a new best solution comes up it could be identified and enforced by ant packets quickly.

Gradient Routing is a routing algorithm specifically conceived for mobile ad hoc networks that takes inspiration form molding. When a node A wants to send a message to B, it actually floods the network with a field-like data structure that holds, the source id, the message and the number of hops from the source of the message to the current node. Such structure not only trivially hands off the message to B (since the message reaches all nodes), but also creates a sort of overlay field leading back to A, to be exploited for further uses. If node B wants to reply, it can just send a message that follows the A field downhill towards A. The field-like distributed data structures created can be used also by other peers to communicate. However, such a field-based data structure has a limited life, in that changes in the

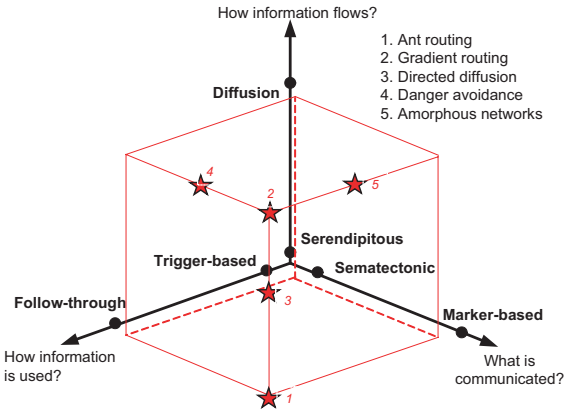

Figure 6. Network approaches in the taxonomy.

network topology due to mobility may invalidate it. The next message sent by A will eventually help in rebuilding it (see Figure 6).

4.5.2. Sensor Networks

Sensor networks are a novel scenario comprising a huge number of micro-sensors wireless interacting to monitor an environment in a coordinated manner [59–62]. The key issues being investigated in the area of sensor networks relate to the identification of effective algorithms and tools to perform distributed monitoring of activities by a cloud of distributed sensors in a physical environment. Representative goals pursued by these researches include tracing the position and movement of an object, determining the occurrence of specific environmental conditions, and reporting sensed data back to a base station in an efficient way. Techniques for self-localization, self-synchronization of activities, and adaptive data distribution, all of which are of primary importance for any type of modern computing scenario, have been widely investigated.

Directed Diffusion [63] is a routing algorithm proposed in the sensor network domain. It makes an explicit use of metaphors like molding and ant foraging. Here the problem is how to collect sensed information from a vast network of sensors dispersed in an environment. The idea is that a workstation at the edge of the network can inject a field-like data structure (inspired to the molding idea) expressing an interest for a particular set of events ("query" field). Sensors are able to route relevant sensed information back to the workstation, exploiting the field data structure as a guide with a mechanism similar to the one described above for the Gradient Routing. Also, the workstation can reinforce some part of the fields, applying a sort of reinforcement learning algorithm similar to ants' pheromone reinforcement. This enables those sensors that have acquired "good" information to report back to the workstation, while allowing others to forget about the "query" field (and thus save battery energy) if they are not able to provide relevant information.

In [64] the goal is to guide a human user or a robot across an environment where a sensor network has been deployed. Sensors can detect dangers nearby and alert and guide users to stay away from dangerous areas. This "avoid danger" navigation has been implemented over a grid of Mica Motes sensors [65]. In particular, a sort of field-based approach — inspired to the molding metaphor — has been conveniently used. In such a potential fields approach, users move under the actuation of artificial forces. The goal (i.e., the intended destination) generates an attractive potential field pulling the user to it. The recognized obstacles generate a repulsive potential which pushes the user away from them. The (negated) gradient of the total potential is the artificial force acting on the user. The direction of this force at a point in the landscape is the current best direction of motion from that point.

4.5.3. Amorphous Computing

The amorphous computing project [66,67] focuses on a visionary scenario in which a massive numbers of identically programmed and locally interacting computational particles are randomly dispersed on a surface or mixed throughout a volume. They are possibly capable of sensing and affecting the environment. Particles have limited resources and local information, and are subject to faults. All particles are programmed identically, although each of them has an autonomous

thread of execution, is capable of generating autonomously random numbers, and has a local state that can depend on past actions. Particles can communicate with each other by short-distance radio connections, or through the substrate itself [67], or by emission of chemicals [68]. The key characteristic of amorphous computing is that particles have no a priori knowledge about the topology of the resulting communication network. No centralized source of information is available, and there are no global clocks, and no global beacons for triangulating positions.

The research in amorphous computing is heavily inspired by biological systems, in particular by the morphogenesis metaphor. In the research on Amorphous Computing, a main mechanism in morphogenesis is the morphogen gradient that resembles a distributed data structure that propagates through a network by changing its values as it propagates. It can be reproduced in an amorphous network of computational particles and it can be used for a variety of purposes related to self-organization of activities. In a network of undifferentiated particles, an initial "source" particle creates a gradient by sending a message to its local neighborhood with the morphogen name and a value of zero. The neighboring particles forward the message to their neighbors with the value increased by one, until the morphogen has propagated through the entire population. Each particle stores and forwards only the minimum value it has received for a particular morphogen name; thus the morphogen value represents the shortest path from the source. By limiting the maximum value of a morphogen, one can create regions of controlled size. The morphogen can also be used for local orientation; a particle can compare values in its local neighborhood to determine the direction towards or away from the source. More than one particle could be the source for the same morphogen, in which case the morphogen value reflects the shortest distance to *any* of the sources. If a single particle emits a morphogen then the value increases as one moves radially away from the particle, but if a line of particles emits a morphogen then the value increases as one moves perpendicularly away. Complex spatial patterning can be created by positioning without any change to the particle program. Since the particles can selectively choose which morphogens to propagate, particles can act as barriers to specific morphogens.

5. Conclusion

This paper introduced a taxonomy of self-organized metaphors and surveyed several applications that could benefit from the use of self-organization. We have only scratched the surface of what self-organization and nature-inspired computing can do to computer science. Self-organization appears to be the next "big concept" in science in general, with applications not only in computer science but in civil engineering, traffic control, medicine, to name but a few. In some of these fields we have already seen results such as buildings inspired by nature formations (what is being called *Bionic Buildings*) [69] and prototypes of intelligent control for vehicles [70] which could have been just easily taken from a flocking metaphor. In computer science the applications abound as this paper demonstrated. However, to allow further progress in this field, one may need to move from an ad-hoc approach to implementing self-organization to more structured and engineered approaches. We believe our taxonomy contributes to this goal. Still, one may require the next step in the development process which is the study of methodologies and programming paradigms that reflect well the self organization of the metaphors described here.

Programmers and designers will understand that in "self-organized design", they are more like conductors of an orchestra — they can set the pace, but in the end, the quality of the orchestra depends on the interactions between the musicians. In computer science, we are too accustomed to be in control and to have supreme power over our designs and implementations. In a self-organized approach, this is not possible given the complexity of the applications. Computer scientists must have a different state of mind and accepting uncertainty in their projects.

This paper contributes to the start of this new way of thinking in computer science where we learn to design with uncertainty but we can be

(probabilistically) confident that the system – at the global scale – will behave as required. We may be seen the birth of a new period in computer science — the self-organized period.

REFERENCES

1. Wikipedia, Self Organization, http://en.wikipedia.org/wiki/Self-organization.
2. A.-L. Barabási, Linked: The New Science of Networks, Perseus Publishing, 2002.
3. M. Buchanan, Nexus: Small Worlds and the Groundbreaking Science of Networks, 1st Edition, W. W. Norton & Company, 2002.
4. H. Rheingold, Smart Mobs: The Next Social Revolution, Perseus Books Group, 2002.
5. F. Vertosick, The Genius Within: Discovering the Intelligence of Every Living Thing, 1st Edition, Harcourt, 2002.
6. S. Johnson, Emergence: The Connected Lives of Ants, Brains, Cities, and Software, Scribner, 2002.
7. S. Strogatz, Sync: The Emerging Science of Spontaneous Order, hyperion Press, 2003.
8. S. Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford Press, 1993.
9. E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Santa Fe Inst. Studies in the Sc. of Complexity Series, Oxford Press, 1999.
10. J. Kennedy, R. C. Eberhart, Swarm Intelligence, Morgan Kaufmann, 2001.
11. S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraula, E. Bonabeau, Self-Organization in Biological Systems, Princeton Univ Press, 2003.
12. M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, 2004.
13. S. Brueckner, Return from the Ant: Synthetic Ecosystems for Manufacturing Control, Ph.D. thesis, Humboldt-Universität zu Berlin, Dept. of Computer Science (2000).
14. D. Gelernter, N.Carriero, Coordination Languages and Their Significance, Communication of the ACM 35 (2) (1992) 96–107.
15. H. Parunak, Go to the Ant: Engineering Principles from Natural Multi-Agent Systems, Annals of Operations Research 75 (1997) 69–101.
16. M. Resnick, Turtles, Termites, and Traffic Jams, MIT Press, 1994.
17. P. Lawrence, The Making of a Fly: the Genetics of Animal Design, Blackwell Science, 1992.
18. C. Reynolds, Flocks, herds and schools: A distributed behavioral model, Computer Graphics 21 (1987) 25–34.
19. J. Toner, Y. Tu, Flocks, herds and schools: A quantitative theory of flocking, Physical Review E. 58 (1999) 4828–4858.
20. C. Breder, Equations Descriptive of Fish Schools and other Animal Agregations, Ecology 35 (1954) 361–370.
21. G. Mead, Mind, Self, and Society, University of Chicago Press, 1934.
22. M. J. Litzkow, M. Livny, M. W. Mutka, Condor: A Hunter of Idle Workstations, in: Proc. of the 8th International Conference on Distributed Computing Systems, 1988.
23. I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, Int. Journal of Supercomputer Applications 11 (2) (1997) 115–128.
24. N. H. Kapadia, J. A. B. Fortes, PUNCH: An architecture for Web-enabled wide-area network-computing, Cluster Computing 2 (2) (1999) 153–164.
25. V. Mann, M. Parashar, Engineering an Interoperable Computational Collaboratory on the Grid, Concurrency and Computation: Practice and Experience 14 (13-15) (2002) 1569–1593.
26. A. Andrzejak, S. Graupner, V. Kotov, H. Trinks, Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems, Tech. Rep. HPL-2002-259, Hewlett-Packard Labs Palo Alto (2002).
27. K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, J. Waldo, The Jini specification, Addison-Wesley, Reading, MA, USA, 1999.
28. G. T. Ltd., GigaSpaces platform, White Paper (February 2002).
29. N. Carriero, D. Gelernter, The S/Net's Linda Kernel, ACM Trans. Comput. Syst. 4 (2)

(1986) 110–129.
30. R. Tolksdorf, R. Menezes, Using Swarm Intelligence in Linda systems, in: A. Omicini, P. Petta, J. Pitt (Eds.), Proc. of the Fourth International Workshop Engineering Societies in the Agents World ESAW'03, 2003.
31. R. Menezes, R. Tolksdorf, A New Approach to Scalable Linda-systems Based on Swarms, in: Proc. of ACM SAC, 2003, pp. 375–379.
32. A. Charles, R. Menezes, R. Tolksdorf, On the Implementation of SwarmLinda, in: Proc. of the ACM Southeastern Conference 04, 2004, pp. 296–297.
33. R. Menezes, R. Tolksdorf, Adaptiveness in Linda-based Coordination Models, in: Proc. of the First International Workshop on Engineering Self-Organising Applications (ESOA 2003), no. LNCS 2977, Springer, 2004.
34. J. Wang, A Survey of Web Caching Schemes for the Internet, ACM Computer Communication Review 25 (9) (1999) 36–46.
35. S. Floyd, Adaptive Web Cache, http://www.icir.org/floyd/web.html (2005).
36. M. Mamei, F. Zambonelli, L. Leonardi, Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination, IEEE Pervasive Computing 3 (2) (2004) 52–61.
37. M. Mamei, F. Zambonelli, Programming Pervasive and Mobile Computing Applications with the TOTA Middleware, in: Proc. of the International Conference On Pervasive Computing (Percom), IEEE CS Press, 2004.
38. A. Hanushevsky, M. Nowark, Pursuit of a Scalable High Performance Multi-Petabyte Database, in: Proc. of the 16th IEEE Symposium on Mass Storage Systems, IEEE Press, 1999, pp. 169–175.
39. M. T. Ozsu, P. Valduriez, Readings in Distributed Computing Systems, IEEE Computer Society Press, 1994, Ch. Distributed Data Management: Unsolved Problems and New Issues, pp. 512–544.
40. G. Weikum, A. Mönkeberg, C. Hasse, P. Zabback, Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering., in: Proc. of the 28th Very Large Databases Conference (VLDB), 2002, pp. 20–31.
41. D. Dasgupta (Ed.), Artificial Immune Systems and Their Applications, Springer-Verlag, 1999.
42. J. O. Kephart, A Biologically Inspired Immune System for Computers, in: Artificial Life IV: Proc. of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, US, 1994, pp. 130–139.
43. M. Sloman, Management Issues for Distributed Services, in: Proc. of the IEEE Second International Workshop on Services in Distributed and Networked Environments, IEEE Press, Whistler, Canada, 1995, pp. 52–59.
44. R. Menezes, Self-Organization and Computer Security: A Case Study in Adaptive Coordination, in: ACM Symposium on Applied Computing, Santa Fe, NM, USA, 2005.
45. Y. U. Cao, A. S. Fukunaga, A. B. Kahng, Cooperative Mobile Robotics, Autonomous Robots 4 (1997) 7–27.
46. O. Khatib, Real-time Obstacle Avoidance for Manipulators and Mobile Robots, Journal of Robotics Research 5 (1) (1986) 90–98.
47. A. Howard, M. Mataric, G. Sukhatme, An Incremental Self-Deployment Algorithm for Mobile Sensor Networks, Autonomous Robots 13 (2) (2002) 113–126.
48. J. McLurkin, J. Smith, Distributed Algorithms for Dispersion in Indoor Environments using a Swarm of Autonomous Mobile Robots, in: Proc. of the International Symposium on Distributed Autonomous Robotic Systems, Toulose, France, 2004.
49. V. Parunak, S. Brueckner, J. Sauter, Digital Pheromones for Coordination of Unmanned Vehicles, in: Proc. of the Workshop on Environments for Multi-agent Systems, LNAI 3374, Springer Verlag, 2004.
50. C. Jones, M. Mataric, From Local to Global Behavior in Intelligent Self-Assembly, in: Proc. of the Conference on Robotics and Automation, IEEE Press, Taipei, Taiwan, 2003.
51. Cooperative Multi-Robot Control Architecture, http://www.dynamic-concepts.com.
52. H. Bojinov, A. Casal, T. Hogg, Emergent Structures in Modular Self-Reconfigurable

Robots, in: Proc. of the Int. Conference on Robotics and Automation, IEEE CS Press, San Francisco, California, USA, 2000.
53. W. Spears, D. Gordon, Using artificial physics to control agents, in: Int. Conference on Information, Intelligence, and Systems, IEEE CS Press, Rockville, Maryland, USA, 1999.
54. M. Mamei, M. Vasirani, F. Zambonelli, Experiments of Morphogenesis in Swarms of Simple Mobile Robots, Journal of Applied Artificial Intelligence 18 (9-10) (2004) 903–919.
55. K. Stoy, R. Nagpal, Self-Reconfiguration Using Directed Growth, in: 7th International Symposium on Distributed Autonomous Robotic Systems, Springer-Verlag, Toulouse, France, 2004.
56. E. Sahin, T. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, M. Dorigo, SWARM-BOTS: Pattern Formation in a Swarm of Self-Assembling Mobile Robots, in: A. El Kamel, K. Mellouli, P. Borne (Eds.), Proc. of the IEEE Int. Conference on Systems, Man and Cybernetics, Piscataway, NJ: IEEE Press, Hammamet, Tunisia, 2002.
57. G.C.Pettinaro, I. Kwee, L. M. Gambardella, F. Mondada, D. Floreano, S. Nolfi, J.-L. Deneubourg, M. Dorigo, SWARM Robotics: A Different Approach to Service Robotics, in: Proc. of the 33rd International Symposium on Robotics, International Federation of Robotics, Stockholm, Sweden, 2002.
58. R. Poor, Embedded Networks: Pervasive, Low-Power, Wireless Connectivity, phD Thesis, MIT (2001).
59. D. Estrin, D. Culler, K. Pister, G. Sukjatme, Connecting the Physical World with Pervasive Networks, IEEE Pervasive Computing 1 (1) (2002) 59–69.
60. K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, M. Welsh, Sensor Networks for Emergency Response: Challenges and Opportunities, IEEE Pervasive Computing 3 (4) (2004) 16–23.
61. M. Paskin, C. Guestrin, A Robust Architecture for Distributed Inference in Sensor Network, in: Proc. of the International Symposium on Information Processing in Sensor Networks, ACM Press, Los Angeles, California, USA, 2005.
62. K. Pister, Invited Plenary Talk, international Conference on Distributed Computing Systems (2003).
63. C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: A scalable and robust communication paradigm for sensor networks, in: Proc. of the International Conference on Mobile Computing and Networking, ACM Press, Boston, Massachusetts, USA, 2000.
64. Q. Li, M. Rosa, D. Rus, Distributed Algorithms for Guiding Navigation across a Sensor Network, in: Proc. of the Int. Conference on Mobile Computing and Networking, ACM Press, San Diego, Rhode Island, USA, 2003.
65. K. Pister, On the Limits and Applicability of MEMS Technology, defense Science Study Group Report (2000).
66. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, R. Weiss, Amorphous Computing, Comm. of the ACM 43 (5) (2000) 74–82.
67. W. Butera, Programming a Paintable Computer, phD Thesis, MIT (2002).
68. R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, I. Netravali, Genetic circuit building blocks for Cellular Computation, Communications, and Signal Processing, Natural Computing 2 (1) (2003) 47–84.
69. R. Buswell, R. Soar, M. Pendlebury, A. Gibb, F. Edum-Fotwe, A. Thorpe, Investigation of the potential for applying freeform processes to construction, in: Proc. of the 3rd Int. Conference on Innovation in Architecture, Engineering and Construction (AEC), 2005, pp. 141–150.
70. P. A. Ioannou, C. C. Chien, Autonomous Intelligent Cruise Control, IEEE Trans. on Vehicular Technology 42 (4) (1993) 657–672.