Nat Comput
DOI 10.1007/s11047-012-9324-y

Description and composition of bio-inspired design patterns: a complete overview

Jose Luis Fernandez-Marquez · Giovanna Di Marzo Serugendo ·
Sara Montagna · Mirko Viroli · Josep Lluis Arcos

Abstract In the last decade, bio-inspired self-organising mechanisms have been applied to different domains, achieving results beyond traditional approaches. However, researchers usually use these mechanisms in an ad-hoc manner. In this way, their interpretation, definition, boundary (i.e. when one mechanism stops, and when another starts), and implementation typically vary in the existing literature, thus preventing these mechanisms from being applied clearly and systematically to solve recurrent problems. To ease engineering of artificial bio-inspired systems, this paper describes a catalogue of bio-inspired mechanisms in terms of modular and reusable design patterns organised into different layers. This catalogue uniformly frames and classifies a variety of different patterns. Additionally, this paper places the design patterns inside existing self-organising methodologies and hints for selecting and using a design pattern.

Keywords Self-organising systems · Bio-inspired mechanisms · Design patterns

1 Introduction

Nowadays, emergent technologies are providing new communication devices (e.g. mobile or smart phones, PDAs, smart sensors, laptops) that form complex infrastructures not widely exploited due to their requirements such as scalability, real-time responses, or failure tolerance. To deal with these features, a new software tendency is to provide entities in the system with autonomy and pro-activity and to increment the interaction between them. This betting on incrementing interaction and decentralising responsibilities over entities, so-called self-organisation, provides systems with better scalability, robustness, and reduces the computation requirements of each entity.

Self-organising mechanisms usually involve decentralisation (no central entity coordinating the re-organisation of the other system's entities) and locality (individual entities have information about their local neighbourhood, i.e. the list of adjacent nodes, information about or provided by these nodes), but no global information, since it is too costly to maintain it up-to-date. Additionally, computation at the micro-level, i.e. at the level of individual entities, involves the execution of relatively simple rules or commands, compared to the complex results these computations reach when considered at a macro-scale. Key characteristics of these mechanisms are robustness and adaptation to changing environmental conditions. Typical self-organising mechanisms are those using stigmergy, like ant foraging for coordinating behaviour, schooling and flocking for coordinating movements, or gradients based systems (de Castr 2006; Di Marzo Serugendo et al. 2011).

J. L. Fernandez-Marquez (✉) · G. Di Marzo Serugendo
University of Geneva, Battelle, Batiment A, Route de Drize 7,
1227 Carouge, Switzerland
e-mail: joseluis.fernandez@unige.ch

G. Di Marzo Serugendo
e-mail: giovanna.dimarzo@unige.ch

S. Montagna · M. Viroli
Alma Mater Studiorum–Università di Bologna, Via Venezia 52,
47521 Cesena, Italy
e-mail: sara.montagna@unibo.it

M. Viroli
e-mail: mirko.viroli@unibo.it

J. L. Arcos
IIIA-CSIC, Campus UAB, 08193 Bellaterra, Spain
e-mail: arcos@iiia.csic.es

Journal : Large 11047
Article No. : 9324
MS Code :
Dispatch : 21-4-2012
Pages : 25
☐ LE
☑ CP
☐ TYPESET
☑ DISK

Self-organising mechanisms are usually inspired by nature, and in particular, by biological systems . Those systems show appealing characteristics for pervasive scenarios, since they are robust and resilient, able to adapt to environmental changes and able to achieve complex behaviours using a limited set of basic rules (Dressler 2010).

Self-organising mechanisms have already been applied to various domains, usually in an ad hoc manner, with varying interpretations and no defined boundary among the used mechanisms. This paper provides a catalogue of bio-inspired mechanisms for self-organising systems. The mechanisms presented are uniformly described and framed using a software design pattern structure identifying when and how to use each pattern, and describing the relation between the different mechanisms. This catalogue of mechanisms is a step forward to engineering self-organising systems in a systematic way.

2 Related work

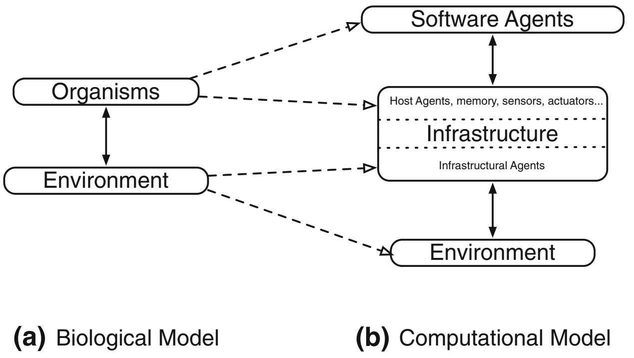
The idea of engineering self-organising systems has attracted many researchers since 2004. Nagpal et al. (2004) present a set of biologically-inspired primitives that describe how organising principles from multi-cellular organisms may apply to multi-agent systems. That paper was a first attempt towards assembling a catalogue of primitives for multi-agent control. However, those primitives are not presented together with an implementation process or by taking into consideration the different scenarios to which the primitives can be applied. It is then difficult to use them in a systematic way for engineering artificial self-organising systems. Mamei et al. (2006) propose a taxonomy to classify self-organising mechanisms and describe a set of mechanisms. These descriptions can drive the implementation of these mechanisms, but they are not expressed as patterns and cannot be used systematically. However, that work motivates to go further and raises new questions: What are the problems that each mechanism can solve? To what solution contributes each pattern? What are the main trade-offs to consider in the implementation? To answer those questions and make the self-organising mechanisms applicable more systematically, some authors have focused on proposing descriptions of self-organising mechanisms under the form of software design patterns (Gamma et al. 1995). The idea of the design pattern structure makes it easy to identify the *problems* that each mechanism can solve, the specific *solution* that it brings, the *dynamics* among the entities and the *implementation*. Gardelli et al. (2007) propose a set of design patterns for self-organising systems all related with the ant colonies behaviour, together with the idea that a mechanism can be composed from other mechanisms. The provided model, however, presents too many constraints to be generalised and the examples of usage are not related to self-organising systems. Based on the set of mechanisms proposed in Mamei et al. (2006), Sudeikat et al. (2008) discuss how intended multi-agent systems (MAS) dynamics can be modelled and refined to decentralised MAS designs, proposing a systematic design procedure that is exemplified in a case study. De Wolf (2007) presents an extended catalogue of mechanisms as design patterns for self-organising emergent applications. The patterns are presented in detail and can be used to systematically apply them to engineering self-organising systems. However, relations among the patterns are missed, i.e. the authors do not describe how patterns can be combined to create new patterns or adapted to tackle different problems.

3 A model to describe bio-inspired design patterns

This section presents the computational model used in this paper to describe the dynamics of the patterns and the relations between the different entities involved in each pattern. The proposed model is clearly inspired by biology but specialised for the artificial world where the patterns will be engineered.

In biological systems, two main entities can be observed: (1) the *organisms* that collaborate in the biological process (e.g. ants, fish, bees, cells, virus, etc.) and (2) the *environment*, a physical space where the organisms are located. The environment provides *resources* that the organisms can use (e.g. food, shelter, raw material) and *events* that can be observed by the organisms and can produce changes in the system (e.g. toxic clouds, storms, thunders, or fires). Organisms can communicate with each other, sense from the environment and act over the environment. Moreover, organisms are autonomous and proactive and they have a partial knowledge of the world. The environment is dynamic and acts over the resources and over the organisms (e.g. it can kill organisms, destroy resources, change the topology of the space where the organisms are living, change the food location, remove food, add new food, etc.). The communication between the organisms can be direct (e.g. dolphins sending ultra-sounds through the water, beavers emitting sounds to alert about a predator presence, etc.) or indirect using the environment to deposit information that other organisms can sense (e.g. pheromone in ants colonies, morphogens in the specialisation of cells, etc).

The biological model may be summarised by two layers: organisms and environment, see Fig. 1a. In order to create a computational model inspired by the biological model, a new layer is added, Fig. 1b. This new layer, called the *infrastructure* layer, is necessary because, in an engineered system, the software agent must be hosted in a device with



(a) Biological Model **(b)** Computational Model

Fig. 1 Relevant entities of the biological and computational models

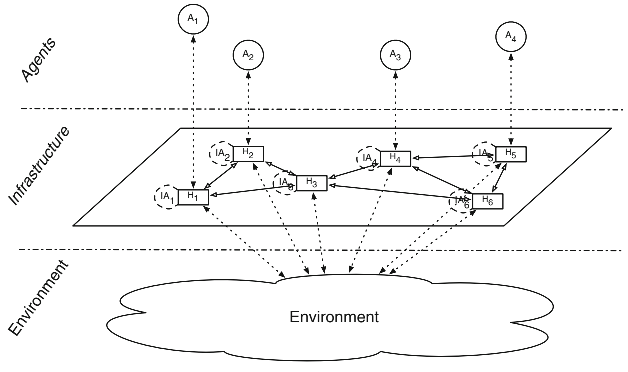


Fig. 2 Computational model

computational power that provides the agents with the ability to interact with the environment (i.e. sensing the environment through sensors or acting in the environment through actuators) and to communicate with other agents.

The entities proposed in the computational model are: (a) the *agents*, that are autonomous pro-active software entities, (b) the *infrastructure*, that contains *hosts* with computational power, sensors and actuators and (c) the *environment*, the real world space where the infrastructure is located. *Events* are phenomena of interest that appear in the environment, can be sensed by the agents using the host's devices. Each agent needs a host to be executed, to communicate with other agents, to sense events or to act in the environment. Thus, the infrastructure provides the agents with all the necessary tools to simulate organisms' behaviour and a place where information can be stored and possibly read by other agents. In most biological processes, the environment plays a key role, due to its ability to act over the entities present in the system (e.g. spreading and removing chemical signals in the environment). To tackle this ability, each host in the infrastructure has an embedded software, called *Infrastructural Agent* (IA). Both IA's and agent's behaviours must be designed to follow self-organising patterns. IAs play an important role when agents can move freely over the hosts. For instance, IAs may be responsible for managing information deposited in hosts by the agents or spreading information over other hosts. In other cases, the IA stands for software embedded into a middleware providing built-in features (e.g. evaporation of digital pheromone).

Figure 2 shows the different layers of the computational model and their corresponding interactions. The top layer represents software agents in the system. Agents use the infrastructure layer to host themselves, communicate with each other, sense and act with the environment and to deposit information that other agents can read. There are two variants in the model: when agents can move freely over the hosts (e.g. mobile agents) or when they are coupled to the host (e.g. swarm of robots). The separation between the agents layer and the infrastructure enables to cover a larger variety of scenarios. On the one hand, software agents may be mobile or may be coupled with hosts. On the other hand the infrastructure may be fixed (i.e. stationary hosts) or mobile. Mobile hosts may be controlled by the agents (e.g. a robot) or not (e.g. PDA's movements under the control of its owner). This is typical of pervasive scenarios where several mobile devices, such as, PDAs, laptops, or mobile phones are located in a common physical space (e.g a shopping mall, a museum, etc.), forming what is usually referred to as an opportunistic infrastructure, where the nodes are moving according to the movements of the user carrying them, and the agents freely jump from one node to another. An example of this architecture is the Hovering Information Project (Fernandez-Marquez et al. 2011), where information is an active entity storing itself and its replica according to some specified spatial structure. Sensor networks are instead a good example of systems where agents are mobile and hosts are not but, on the other hand, they also well represent systems where not only hosts but also agents are static, as reported in (Vinyals et al. 2011).

To summarise, the entities used in the computational model are:

- *Agents* autonomous and pro-active software entities running in a host.
- *Infrastructure* the infrastructure is composed of a set of connected Hosts and Infrastructural Agents. A *Host* is an entity with computational power, communication capabilities and may have sensors and actuators. Hosts provide services to the agents. An *Infrastructural Agent* is an autonomous and pro-active entity, acting over the system at the infrastructure level. Infrastructural Agents may be in charge of implementing those environmental behaviours present in nature, such as diffusion, evaporation, aggregation, etc.
- *Environment* the Environment is the real world space where the Infrastructure is located. An *Event* is a phenomenon of interest that appears in the Environment

and that may be sensed by the Agents using the sensors provided by the Hosts.

In this paper, we regard a system as composed of Agents, Infrastructure, Infrastructural Agents, Hosts, and Environment. The behaviour of Agents and Infrastructural Agents is defined by a set of rules (hereafter referred to as *transition rules*), while Hosts are defined by the interface they provide.

4 Design patterns as part of methodologies for self-organising systems

Current methodologies for self-organising systems (Puviani et al. 2012) follow the typical phases of software engineering methodologies: requirements, analysis, design, implementation, verification and test. Even though these methodologies all put focus on different aspects, they each accommodate a specific design phase where interaction mechanisms are identified, modelled, refined and possibly simulated. Consequently, self-organising design patterns are best exploited during the design phase of a chosen methodology.

The design patterns come into play during the design phase, which we propose to split into three distinct steps (Fig. 3): (1) the choice of design patterns is made during an early phase of design. Self-organising design patterns serve to identify the problem to solve as well as to determine the appropriate solution to bring to the problem. In particular, they help determining the boundaries of each problem and its corresponding solution provided by the pattern; (2) during a refined phase, actual entities and their dynamics are defined. The patterns' dynamics serve to refine the model and to identify the entities and their precise interactions, individual responsibilities and to anticipate the emergent behavior; (3) finally, during the simulation step, the patterns implementation description will serve to establish implementation details in relation with the underlying computational model. These three steps can be iterated in a loop in order to progressively refine or review the model. An important issue with self-organising mechanisms concerns the parameters tuning. Patterns come with a description of the main parameters involved in the pattern and their effect on the resulting behavior. The simulation phase is then crucial for determining the parameters values.

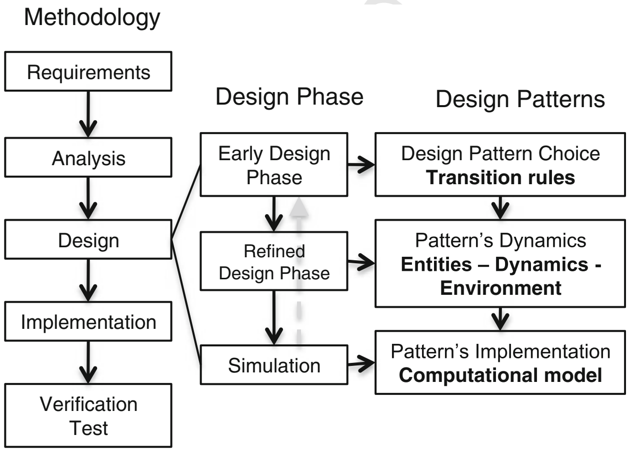


Fig. 3 Design patterns within the design phase of SO methodologies

5 Design patterns' catalogue

To create the patterns' catalogue, we analysed the inter-relations among the self-organising mechanisms for engineering self-systems existing in the literature, in order to understand how they work and to facilitate their adaptation or extension to tackle new problems. The classification process started by selecting those high-level mechanisms that are well-known in the literature and have been applied successfully to different self-* systems. By analysing their behaviours, we identified common lower-level mechanisms, some of them basic (atomic) and other composed of basic ones. As a result, we classified the patterns into three layers. The *basic* mechanisms that can be used individually or in composition to form more complex patterns are at the bottom layer. At the middle layer, there are the mechanisms formed by *combinations* of the bottom layer mechanisms. The top layer contains higher-level patterns that show different ways to *exploit* the basic and composed mechanisms.

Figure 4 shows the different design patterns collected in the catalogue and their relations. The arrows indicate how the patterns are composed. A dashed arrow indicates that it is optional (e.g. the Gradient Pattern can use evaporation, but the evaporation is not necessary to implement gradients).

This classification aims at listing existing mechanisms from the literature, identifying their own boundaries (i.e. when one mechanism stops, and when another starts), their inter-relations and the recurrent problem they solve. For example, Gossip has been applied to many works in different ways, but all implementations share the fact that

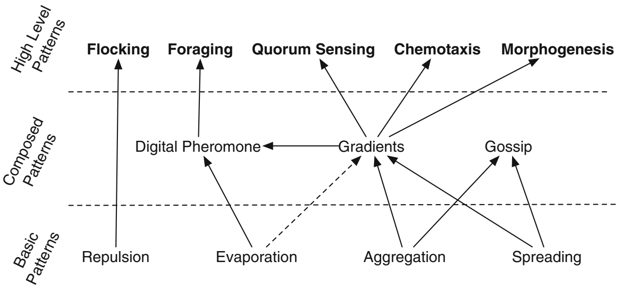


Fig. 4 Patterns and their relationships

gossip is a process composed of the spreading and aggregation mechanisms. The catalogue provided in this paper does not intend to be exhaustive. Instead it is meant to be open to new additions. New basic (atomic) mechanisms can be added to the catalogue once they are identified and described under the form of patterns. Similarly, any new identified combination of basic or higher level patterns can be as well added to the catalogue.

Patterns are described in Table 1. For each pattern, besides its name and other known appellations, the problem it addresses and the solution it provides are clearly identified. Additional fields precise the biological inspiration for the pattern, the effect of key parameters involved in the pattern, the entities involved and their dynamics, as well as environmental requirements. Implementation or simulation descriptions are provided, together with references to known uses in the literature, consequences of the use of the pattern and a list of other patterns that are used by or that exploit the considered pattern.

The behaviour of patterns is described through transition rules using the following simple notation. Each information in the system is modelled as a tuple $\langle L, C\rangle$, where L is the location where the information is stored, and C is its current content, e.g. in the form of a list with one or more arguments of different types, such as numbers, strings or structured data, according to the application specific information content.

Transition rules are chemical-resembling reactions working over patterns of tuples. They are of the kind:

$$\texttt{name} :: \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle \xrightarrow{r} \langle L'_1, C'_1\rangle, \ldots, \langle L'_m, C'_m\rangle$$

where (i) the left-hand side (reagents) specifies which tuples are involved in the transition rule: they will be removed as an effect of the rule execution; (ii) the right-hand side (products) specifies which tuples are accordingly to be inserted back in the specified locations: they might be new tuples, transformation of one or more reagents or even unchanged reagents; and (iii) rate r is a rate, indicating the speed/frequency at which the rule is to be fired, namely, its scheduling policy. Rules are then equipped with a set of transition rules that determine the right-hand side variables as functions of the left-hand side ones. Such functions (including e.g. evaporation slope) may be subject to conditions and constrains, which will be specified together with the reaction. Note that such functions could be:

1. fixed parameters of the system we model;
2. automatically extracted from reagents, e.g. an information item also stores the function it should be applied to; or
3. actually specified in the transition rule.

Our model of transition rules intentionally abstracts from these aspects. As a notational convenience, we will use notation $\{x, y, z, \ldots\}$ for sets, and $(x; y; z; \ldots)$ for ordered sequences.

5.1 Basic patterns

Basic patterns are atomic patterns, used to compose more complex patterns appearing at the middle layer (Sect. 5.2) and at the top layer (Sect. 5.3). These patterns describe basic mechanisms that have been frequently used in the literature.

5.1.1 Spreading pattern

The Spreading Pattern is based on direct communication among agents for progressively sending information over

Table 1 Description fields

Name	The pattern's name
Aliases	Alternative names used for the same pattern
Problem	Which problem is solved by this pattern and situations where the pattern may be applied
Solution	The way the pattern can solve the problems
Inspiration	Biological process inspiring the pattern
Forces	Prerequisites for using the pattern and aspects of the problem that lead the implementation, including parameters (trade-offs)
Entities	Entities that participate in the pattern and their responsibilities. Entities are agents, infrastructural agents, and hosts
Dynamics	How the entities of the pattern collaborate to achieve the goal. A Typical scenario describing the run-time behaviour of the pattern
Environment	Infrastructural requirements of the pattern
Implem./simulation	Hints of how the pattern could be implemented, including parameters to be tuned
Known uses	Examples of applications where the pattern has been applied successfully
Consequences	Effect on the overall system design
Related patterns	Reference to other patterns that solve similar problems, can be beneficially combined or present conflicts with this pattern

the system. The spreading of information in multi-agent systems allows the agents to increment the global knowledge of the system. Figure 5 shows the different steps of the spreading process: (a) an agent initiates the spreading process (black node); (b) the information spreads over the network; and (c) the process finishes when information reaches all the nodes in the network.

Aliases spreading is also known as information diffusion (Khelil et al. 2002), information or data dissemination (Sabbineni 2005), flooding (Yi 2003), broadcast (Tseng et al. 2002), or epidemic spreading (Khelil et al. 2002).

Problem in systems, where agents perform only local interactions, agents' reasoning suffers from the lack of knowledge about the global system.

Solution a copy of the information (received or held by an agent) is sent to neighbours and propagated over the network from one node to another. Information spreads progressively over the system and reduces the lack of knowledge of the agents while keeping the constraint of the local interaction.

Inspiration spreading is a basic pattern extended or exploited by most other patterns presented in this catalogue. Spreading appears in important processes, such as, *Morphogenesis*, *Chemotaxis* or *Quorum Sensing* (Sect. 5.3) In nature, spreading is a process done by the environment.

Forces if spreading occurs with high frequency, the information spreads over the network quickly but the number of messages increases. A quick spread is desired when the environment is continuously changing and the agents must know the new values and adapt themselves. It may happen that the information is only interesting for agents close to the source. In that case, the information spreads only up to a determined number of hops, reducing the number of messages. Another way to reduce the number of messages is to determine the number of neighbouring nodes that receive the information. It was demonstrated that it is not necessary to send the information to all the neighbouring nodes in order to ensure that every node has received the information (Birman et al. 1999).

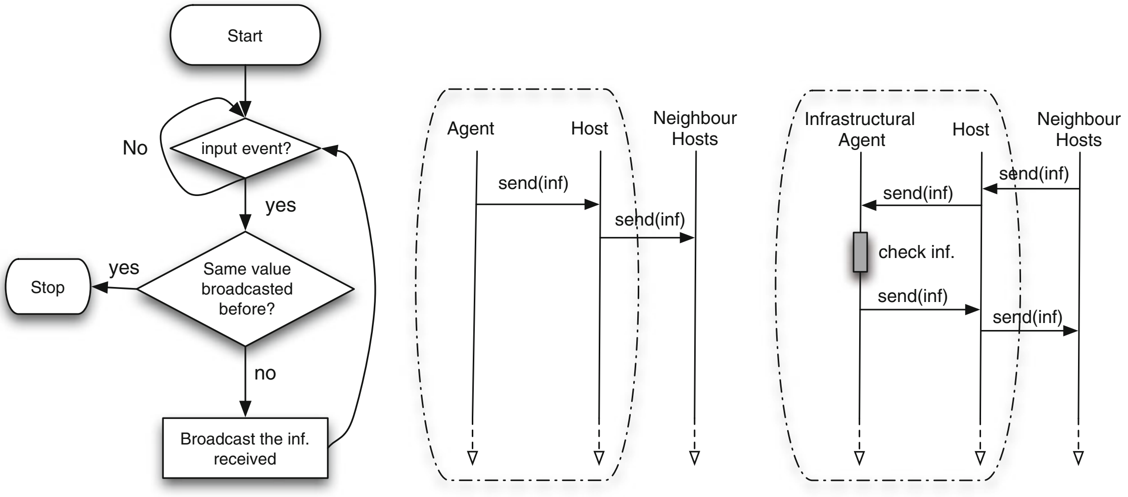
Entities-Dynamics-Environment the entities involved in the spreading process are the hosts, agents, and infrastructural agents. The spreading process is initiated by an agent that first spreads the information in the host it is residing in. When this information arrives to neighbouring nodes, the infrastructural agent is in charge to re-send the information to neighbouring nodes, producing the spreading of the information over the whole system.

Each infrastructural agent forwards the information received to a specified number of neighbours and up to the specified number of hops. The dynamics is usually extended to avoid infinite loops and wasted duplicate deliveries (e.g. when one agent receives the same information it has sent before, the agent does not resend that information).

Transition Rule (1) describes more formally the Spreading Pattern.

$$\begin{aligned} &\texttt{spreading} :: \langle L, C\rangle \xrightarrow{r_{spr}} \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle \\ &\quad \textit{where } (L_1;\ldots;L_n) = \nu(L), (C_1;\ldots;C_n) = \sigma(C, L) \end{aligned} \tag{1}$$

A function $\nu(L)$ is given for determining the sequence of locations, among the neighbours of L, to which the information in input has to be spread. The set of such locations cannot be empty, cannot be composed of L only, but can be composed of all the neighbourhood of L including L itself.



(a) Infrastructural Agent behaviour **(b)** Initialisation **(c)** Interactions

Fig. 5 Spreading: infrastructural agent behaviour (**a**), corresponding initialisation (**b**), and interactions with its host and neighbouring hosts (**c**)

A function $\sigma(C, L)$ is given for computing the new information content, which may change within the spreading process.

Implementation the most common algorithm used to spread the information to the neighbours is the broadcast algorithm.

It is well known that broadcast causes what is called as the Broadcast Storm Problem (Tseng et al. 2002). The Broadcast Storm Problem appears when the radius of the signal of many nodes overlaps. Thus, a straightforward broadcasting by flooding will result in serious redundancy, contention and collision. In order to solve the Broadcast Storm Problem, an optimised broadcast can be implemented, which can follow a probabilistic, counter-base, distance-base, location-base or cluster-base schema (Tseng et al. 2002). As time goes by, new proposals for efficient ways of spreading the information are proposed.

This work presents a basic implementation to illustrate how spreading works and how it has been implemented in the literature. Further comparison between different kinds of spreading implementations and their performances is out of the scope of this work.

Figure 5a shows the flow chart where the information spreads after it is received. Figure 5b shows the interaction diagram of the spreading initialisation. Figure 5c represents the interactions when the information arrives to a neighbour.

Known uses the spreading mechanism has been applied to several applications: Swarm motion coordination (Parunak et al. 2002), coordination in games (Mamei 2004), and problem optimisation (Blu 2005). More references of applications can be found in higher level patterns that exploit the Spreading Pattern (i.e. Gradient Pattern, Morphogenesis Pattern, Chemotaxis Pattern and Quorum Sensing Pattern).

Consequences when the Spreading Pattern is applied, the agents in the system sense information from beyond their local sensing. Then, there is an increment of the network load (i.e. messages and memory). This increment becomes extreme when the environment is very dynamic and the agents have to keep the information updated as soon as possible.

Related Patterns spreading is used in higher level patterns such as Gradient (Sect. 5.2.1), Morphogenesis (Sect. 5.3.3), or Chemotaxis Pattern (Sect. 5.3.2).

5.1.2 Aggregation pattern

The Aggregation Pattern is a basic pattern used for information fusion. The dissemination of information in large scale systems, either deposited by the agents or taken from the environment, may produce network and memory overload. The Aggregation Pattern was introduced as a way to reduce the amount of information in the system by synthesising meaningful information (Gardelli et al. 2007).

Alias aggregation is also known as fusion (Niu 2005).

Problem in large systems, excess of information produced by the agents may produce network and memory overloads. Information must be distributively processed in order to reduce the amount of information and to obtain meaningful information.

Solution aggregation consists in locally applying a fusion operator to process the information and synthesise macro information. This fusion operator can take many forms, such as filtering, merging, aggregating, or transforming (Chen 2002).

Inspiration in nature, the aggregation (sum) of ant's pheromones allows the colony to find the shortest path to the food, and to discard longer paths. (i.e. two pheromone scents together create an attractive field bigger than a single pheromone scent). In nature the aggregation is a process performed by the environment. Even when there are no agents present in the system, the environment continues performing the aggregation process.

Forces aggregation applies to all the information available locally or only on part of that information. The parameter involved is the amount of information that is fused; it relates to the memory usage in the system.

Entities-Dynamics-Environment aggregation is executed either by agents or by infrastructural agents. In both cases the agents aggregate the information they access locally. Information may come from the environment or from other agents. Information coming from the environment is typically read by sensors (e.g. temperature, humidity, etc.). According to the model presented in Sect. 3, aggregation is executed by an agent that receives information from the host where the agent is residing. Such host is either a sensor reading information from the environment or a communication device receiving information from neighbouring hosts. Aggregation may be applied by any agent that receives information independently of the underlying infrastructure. The aggregation process is not repetitive and finishes when one agent executes the aggregation function.

The Transition Rule for aggregation (2) is as follows: information in input (possibly a set of information) is transformed into a new set of information with smaller cardinality then the input set through an aggregation function α.

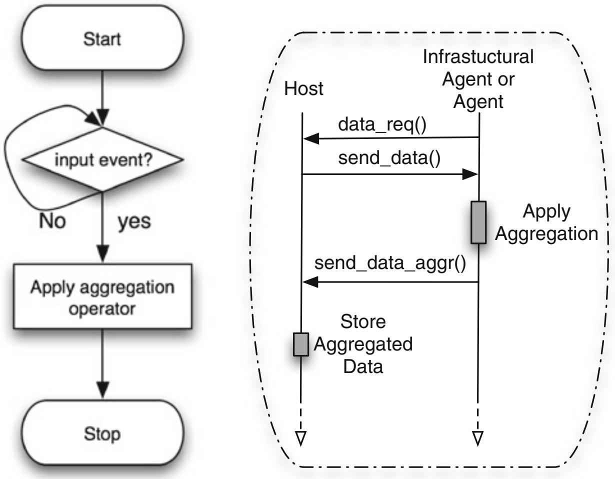
$$\texttt{aggregation} :: \langle L, C_1\rangle, \ldots, \langle L, C_n\rangle \xrightarrow{r_{aggr}} \langle L, C'_1\rangle, \ldots, \langle L, C'_m\rangle$$

$$\textit{where } \{C'_1, \ldots, C'_m\} = \alpha(\{C_1, \ldots, C_n\}) \qquad (2)$$

Implementation available information takes the form of a stream of events. Aggregation or fusion of information can take various forms: from a simple operator (sum, multiplication or average) like in ACO, to more complex operators (e.g. Kohonen Self-Organising Maps aggregating sensor data in clusters, Lee 2004). Fusion operators are classified into four different groups (Chen 2002): (1) *filter*: this operator selects a subset of the received events (e.g. the sensor takes 10 measures per second, but the application processes only 1 per second); (2) *transformer*: this operator changes the type of the information received in input (e.g. inputs are GPS coordinates and outputs are the countries where the positions are located); (3) *merger*: this operator unifies all information received and outputs all information received as a single piece of information (e.g. input is the position of many sensors and the output is the corresponding tuple of positions); (4) *aggregator*: this operator applies a specific operation (e.g. max, min or avg) to one or more incoming information; input and output types can all be different. The flow chart 6a shows that the aggregation process starts when the agent receives the information (an event). Then, it applies the fusion operator and sends the aggregated information back to the host. Figure 6b shows how the agent or infrastructural agent uses the interface provided by the host to get the data, applies a fusion operator, and deposits the aggregated data back in the host.

Known uses aggregation has been used in the ACO algorithm (Dorigo 1999) to aggregate pheromones, emulating higher concentrations when two or more pheromones are close to each other. Aggregation is also used in digital pheromones for autonomous coordination of swarming UAVs (Parunak et al. 2002). Moreover, aggregation has been used in the field of information fusion, which studies how to aggregate individual belief bases into a collective one (Grégoire 2006), or for truth-tracking in MAS (Pigozzi 2007).

Consequences aggregation increases the efficiency in networks (e.g. sensor networks, ad-hoc or P2P), by reducing the number of messages, i.e. increasing the battery life and the scalability of the system. Also aggregation provides a mechanism to extract macro-information in large-scale systems, such as extracting meaningful information from data reads obtained from many sensors. Thus, the amount of memory used by the system is reduced.



(a) Flow **(b)** Interaction

Fig. 6 Aggregation: agent behaviour

Related Patterns the Aggregation Pattern can be implemented together with Evaporation and Gradient Patterns to form digital pheromones (Parunak et al. 2002). Evaporation can be used with aggregation in order to aggregate information recently collected from the environment. The Gossip Pattern (Sect. 5.2.3) is a pattern composed of the Aggregation Pattern and the Spreading Pattern (Sect. 5.1.1).

5.1.3 Evaporation pattern

Evaporation is a pattern that helps dealing with dynamic environments where information used by agents can become outdated. In real world scenarios, the information appears and changes with time and its detection, prediction, or removal is usually costly or even impossible. Thus, when agents have to modify their behaviour taking into account information from the environment, information gathered recently must be more relevant than information gathered a long time ago. Evaporation is a mechanism that progressively reduces the relevance of information. Thus, recent information becomes more relevant than information processed some time ago. Evaporation was proposed as a design pattern for self-organising multi-agent systems in (Gardelli et al. 2007) and is usually related to Ant Colony Optimisation (ACO) (Dorig 1992).

Aliases evaporation is also known as decay (Huebel et al. 2008), temporal degradation function (Ye et al. 2008) or freshness (Ranganathan et al. 2004).

Problem outdated information cannot be detected and it needs to be removed, or its detection involves a cost that needs to be avoided. Agent decisions rely on the freshness of the information presented in the system, enabling correct responses to dynamic environments.

Solution evaporation is a mechanism that periodically reduces the relevance of information. Thus, recent information becomes more relevant than older information.

Inspiration evaporation is present in nature. For instance, in ant colonies (Deneubourg et al. 1983), when ants deposit pheromones in the environment, these pheromones attract other ants and drive their movements from the nest to the food and vice-versa. Evaporation acts over the pheromones reducing their concentration along the time until they disappear. This mechanism allows the ants to find the shortest path to the food, even when environment changes occur (such as, new food locations or obstacles in the path). Ants are able to find the new shortest paths by discarding the old paths.

Forces evaporation is controlled by the parameters evaporation factor (i.e. how much the information is evaporated) and the evaporation frequency (i.e. frequency of evaporation execution), used to decrement the relevance of the information. The evaporation factor and evaporation frequency must deal with the dynamics of the environment: if evaporation is too fast, we may lose information; if evaporation is too slow, the information may become outdated and misguide the agents' behaviour. A higher evaporation factor releases memory, but also reduces the information available in the system for the agents. When the evaporation is applied to collaborative search or optimisation algorithms, the evaporation factor controls the balance between exploration and exploitation: high evaporation rates reduce agents' knowledge about the environment, increasing the exploration, and producing fast adaptation to environment changes. However, a higher evaporation factor decreases the performance when no environment changes occur (due to an excess of exploration).

Entities-Dynamics-Environment evaporation can be applied to any information present in the system. Periodically, its relevance decays over time. Thus, recent information becomes more relevant than information processed some time ago.

Evaporation is performed by the agent or infrastructural agent periodically executing Transition Rule (3).

$$\begin{aligned}&\texttt{evaporation} :: \langle L, C\rangle \xrightarrow{r_{ev}} \langle L, C'\rangle\\&\textit{where } C' = \epsilon(C)\end{aligned} \tag{3}$$

The rule affects the relevance value contained in C applying the function ϵ that can, for instance, impose that $Rel_{C'} = Rel_C * Ev_{\text{factor}}$ with $Ev_{\text{factor}} \in [0, \ldots, 1]$ or that $Rel_{C'} = Rel_C - Ev_{\text{factor}}$. The requirement for $\epsilon(C)$ is that the relevance value decreases with the application of the rule.

Implementation the Evaporation Pattern is executed by an agent that needs to update the relevance of its internal information, or by infrastructural agents that change the relevance of the information deposited in an environment. We distinguish two approaches. In the first approach, an agent encapsulates the information and decays its own relevance. In this case, the agent follows the flow chart 7a and the corresponding interaction diagram 7b. In the second approach, the information is deposited by one agent in a host and an infrastructural agent interacts with the host to decay the information's relevance. The host provides an interface for reading and changing the relevance value. In this case, the interaction between the infrastructural agent and the host is shown in Fig. 7c.

Known uses evaporation has been used mainly in Dynamic Optimisation. Examples of algorithms using evaporation are ACO (Dorigo 1999) and Quantum Swarm Optimisation Evaporation (QSOE) (Fernandez-Marquez 2009). In some other works, evaporation is performed using a parameter called freshness associated to the information (Weyns et al. 2006).

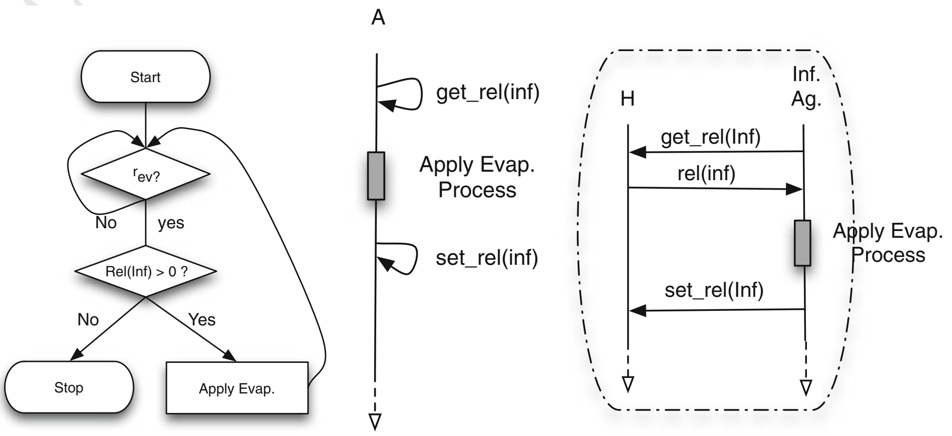
Consequences evaporation enables adaptation to environmental changes. However, the use of evaporation in static scenarios may decrease the performance, due to the loss of information associated to this mechanism. The Evaporation Pattern provides the ability of self-adapting to environmental changes increasing the tolerance to noise, as shown in (Fernandez-Marquez 2010).

Related Patterns the Evaporation Pattern is used by higher level patterns such as Digital Pheromone Pattern (Sect. 5.2.2) or Gradient Pattern (Sect. 5.2.1).

5.1.4 Repulsion pattern

The Repulsion Pattern is a basic pattern for motion coordination in large scale MAS. The Repulsion Pattern enables

Fig. 7 Evaporation: agent behaviour (**a**), evaporation by the agent itself (**b**), evaporation by the host (**c**)



(**a**) Agent flow (**b**) Agent (**c**) Agent - Host the agents to get a uniform distribution in a specific area or to avoid collision among them. Moreover, using repulsion, agents can adapt their position when the desired area changes or when some nodes disappear.

Alias none to our knowledge.

Problem agents' movements have to be coordinated in a decentralised manner in order to achieve a uniform distribution and to avoid collisions among them.

Solution the Repulsion Pattern creates a repulsion vector that guides agents to move from regions with high concentrations of agents to regions with lower concentrations. Thus, after few iterations agents reach a more uniform distribution in the environment.

Inspiration the repulsion mechanism appears in a wide range of biological self-organising processes, such as the diffusion process in physical science, the flocking of birds or schools of fish. For instance, the diffusion process describes the spread of particles through random motion from regions of higher concentration to regions of lower concentration. Figure 8 illustrates the different steps of the diffusion process. First, a concentration of ink is deposited in the glass of water, step (a). We observe the initial state where the particles concentrate in one corner of the glass. The corner with the particles, therefore, contains a higher concentration of ink's particles. Second, the particles begin to move in the diffusion process, from regions of higher concentration to regions of lower concentration, step (b). The closer the particles are to the corner, the higher the concentration, thus creating a so called concentration gradient. This gradient is provided by the difference in concentration between neighbouring particles. Finally, we observe how the diffusion process has randomly moved around all the particles inside the water, producing a uniform random distribution of the particles. At this point the different ink's concentrations disappear. Inside a container, the particles reach a uniform distribution after the diffusion process. However, in an open space, the diffusion process spreads the particles until the concentration is so low that it is considered negligible. As Fig. 8 shows, the diffusion process finishes when the particles reach a uniform distribution, i.e. when the concentration gradient becomes zero. The repulsion mechanism is also alternatively presented as inspired by the gas theory (Cheng et al. 2005). In the case of gas theory, the time to reach a uniform concentration is shorter than in the case of the diffusion process.

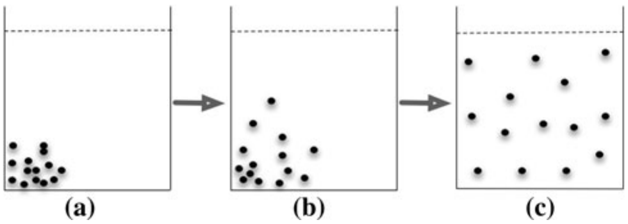


Fig. 8 Diffusion in science

Forces the main parameters involved in the Repulsion Pattern are the repulsion frequency (i.e. how frequent the repulsion is applied) and the repulsion radius (i.e. how strong the repulsion is). A high repulsion frequency involves a faster spreading of the agents and a faster adaptation when the desired formation (or area) changes. However, it increases the number of messages, because the Repulsion Pattern requires information about the position of neighbours. The repulsion radius should be limited to the communication range of the agents, because it makes not sense to move to one location where the concentration of agents is unknown and also because the agent can not jump to a host that is not in the communication radius. Thus, the movement of one agent in each repulsion step must be restricted to its communication range.

Entities-Dynamic-Environment repulsion can be applied in systems where the agents are residing in mobile hosts (e.g. robotic swarms) or in software agents that are moving freely in a network composed of (stationary or not) hosts. In both cases the dynamics between them is the same. When repulsion is applied, the agent that executes the repulsion sends a position request to all its neighbouring agents. After the agent receives the positions of neighbouring hosts, it calculates the desired position and moves to that position. When the environment is not continuous, as in the mobile agents case, the agent moves to the host closest to the desired position. In this case the position request must be sent also to the hosts.

To apply the Repulsion Pattern, each agent should know its position and its neighbourhood. The Repulsion Pattern may apply also to information that might need to be spatially distributed.

Transition Rule (4) precises the repulsion behaviour:

$$\begin{aligned}&\texttt{repulsion} :: \langle L, C\rangle, \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle \xrightarrow{r_{ev}}\\ &\quad \langle L', C\rangle, \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle\\ &\quad \textit{where } L' = \rho(\{\langle L, C\rangle, \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle\})\end{aligned} \tag{4}$$

A function $\rho(\{\langle L, C\rangle, \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle\})$ is given for computing the new location of the information or of the agent according to the spatial distribution of the neighbours and to its actual position. An example of such a function follows. Function ρ depends also on the values of attributes contained in C, for instance the concentration of particles in each location.

Implementation one possible implementation to reach a uniform distribution, involves a transition rule that calculates a repulsion vector between the particles that is inversely proportional to the distance between them. The transition rule is then implemented as follows: Let R be the repulsive radius; d_i the distance between a given node and neighbouring node i; p the position of the given node and p_i the position of the neighbouring node i. Then, the position p_{t+1} of the agent at time $t+1$ and the movement vector m are given by:

$$p_{t+1} = p_t + \mathbf{m} \tag{5}$$

$$\mathbf{m} = \sum_i \frac{\mathbf{p} - \mathbf{p_i}}{d_i}(R - d_i) \tag{6}$$

Figure 9 shows how agent 1 is repelled by agents 2 and 3 when it applies the repulsion mechanism. In Fig. 9a agent 1 executes Eq. (6) to create the repulsion vector. In Fig. 9b agent 1 moves by following the repulsion vector.

Figure 10a shows the behaviour of an agent that is executing the Repulsion Pattern. At the beginning the agents send a position request to all the agents in the communication range. When positions are received, the repulsion vector is calculated following Eq. (6) and then, the new desired position by using Eq. (5). At this step if the system is composed of a swarm of robots, the robot that is executing the Repulsion Pattern would move to the desired position. If the Repulsion Pattern is executing using a mobile agents technology, the agent would move to the closest node to the desired position. Figure 10b shows the interaction between the agent that is executing the Repulsion Pattern, the host where the agent is running and their neighbouring hosts.

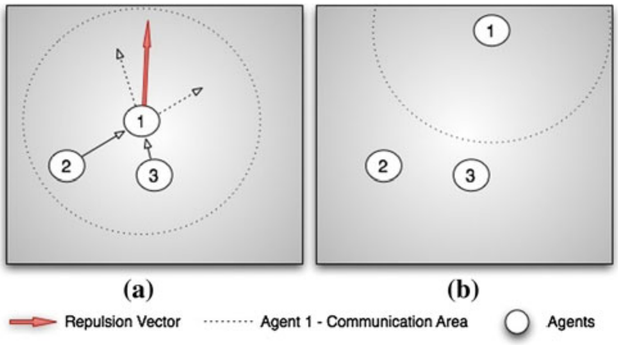


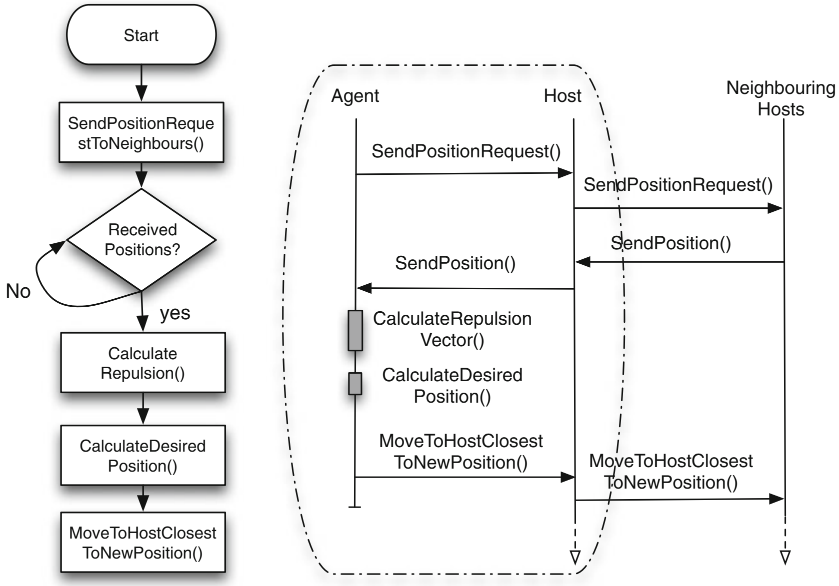
Fig. 9 Repulsion

Known uses repulsion has not been proposed as a pattern so far. Several applications have used the repulsion mechanism, such as swarm robotics for pattern formation (Cheng et al. 2005), where the system achieves shape formation by simultaneously allowing agents to disperse within a defined 2D shape. In Particle Swarm Optimisation (PSO), Repulsion coordinates the position of explorer particles in a multi-swarm approach (Fernandez-Marquez 2009). In (Fernandez-Marquez et al. 2011), the repulsion is used to coordinate the position of pieces of information, ensuring the accessibility to this information in a specific area of interest using the minimum possible memory.

Consequences repulsion does not involve replication, i.e. during the repulsion process no new agents are created, contrarily to spreading. Repulsion is a continuous process that produces a uniform distribution of the agents in the system. Even when the agents are uniformly distributed in the environment, the repulsion mechanism continues working, producing a self-adaptation process when the number of agents changes (i.e. self-repairing formation in swarms of robots) or environmental changes occur.

Related Patterns the Repulsion Pattern is used in the Flocking Pattern (Sect. 5.3.5).

Fig. 10 Repulsion: agent behaviour



(a) Flow **(b)** Interactions

5.2 Composed patterns

This section analyses compositions of basic patterns, widely used in the literature. It provides composed patterns that can be used on their own or extended in turn by higher level patterns.

5.2.1 Gradient pattern

The Gradient Pattern is an extension of the Spreading Pattern where the information is propagated in such a way that it provides an additional information about the sender's distance: either a distance attribute is added to the information; or the value of the information is modified such that it reflects its concentration - higher concentration values meaning the sender is closer, such as in ants' pheromones. Additionally, the Gradient Pattern uses the Aggregation Pattern to merge different gradients created by different agents or to merge gradients coming from the same agent but through different paths. Different cases may apply: either only the information with the shortest distance to the sender is kept, or the concentration of the information increases.

Aliases the Gradient Pattern is a particular kind of computational fields (Bea 2009) (i.e. physical fields based abstractions).

Problem agents belonging to large systems suffer from lack of global knowledge to estimate the consequences of their actions or the actions performed by other agents beyond their communication range.

Solution information spreads from the location it is initially deposited and aggregates when it meets other information. During spreading, additional information about the sender's distance and direction is provided: either through a distance value (incremented or decremented); or by modifying the information to represent its concentration (lower concentration when information is further away). Thus, agents that receive gradients have information that come from beyond their communication range, increasing the knowledge of the global system not only with gradients information but also with the direction and distance of the information source. During the aggregation process, a filter operator keeps only the information with the highest (or lowest) distance, or it modifies the concentration. Gradients can deal with network topology changes. In this case the information spreads periodically and is subject to evaporation, reducing its relevance along the time, and enabling the gradients to adapt to networks topology changes. Such gradients are called active gradients (Clement 2003).

Inspiration gradients appear in many biological processes. The most known are Ant Foraging, Quorum Sensing, Morphogenesis, and Chemotaxis processes. In these processes, gradients support long-range communication among entities (cells, bacteries, etc..) through local interaction.

Forces adaptation to environmental changes is faster when updating frequencies are high, thus increasing network overload. Lower updating frequencies reduce network overload, but can lead to outdated values when environmental changes occur. There is a trade-off between the diffusion radius (number of hops) and the load in the network. A higher diffusion radius brings information further away from its source, providing guidance to distant agents. However, it increments the load and may overwhelm the network (Bea 2009).

Entities-Dynamic-Environment entities acting in the Gradient Pattern are Agents, Hosts, and Infrastructural Agents. Analogously to the Spreading Pattern, when a gradient is created, it is spread to its neighbours.

The transition rules for the Gradient Pattern are specific instances of Transition Rule (1) and Transition Rule (2). An example is given in Transition Rules (7). We assume that each tuple contains a D attribute that represent the distance from the current host to the source of the gradient.

$$\begin{aligned}
&\texttt{spreading} :: \langle L, [D, C]\rangle \xrightarrow{r_{spr}} \langle L_k, [D \pm \Delta D, C]\rangle \\
&\quad \mathit{where}\ L_k = \mathit{random}(\{L_1, \ldots, L_n\}) \\
&\texttt{aggregation} :: \langle L, [D_1, C]\rangle, \ldots, \langle L, [D_n, C]\rangle \xrightarrow{r_{aggr}} \langle L, [D', C]\rangle \\
&\quad \mathit{where}\ D' = \mathit{min}/\mathit{max}(\{D_1, \ldots, D_n\}) \qquad (7)
\end{aligned}$$

The first transition rule models the spreading of information modifying the distance attribute by incrementing or decrementing its value so to get to a cone-shaped gradient with the vertex down or up. Moreover, the rule specifies a specific instance of the function $v(L)$ introduced in Transition Rule (1) for determining the sequence of locations, among the neighbours of L, to which the information in input has to spread. Such a function $\mathit{random}(\{L_1, \ldots, L_n\})$ chooses randomly one location among all the neighbouring locations of L. The second transition rule models the corresponding case of aggregation when multiple tuples with the same content but different distance attribute are locally present. This particular rule models the case of an aggregation where only the information with the shortest / longest distance is kept. It is important to note that D could also represent concentrations instead of distances.

Implementation agents start the process by sending information to all their neighbours, as shown in Fig. 11 b for the case with distance value. When one agent receives the information it increments the distance attribute, or it reduces accordingly the concentration value of the information, and forwards the gradient again to all its neighbours (Spreading Pattern) as shown on diagram flow Fig. 11a and sequence diagram Fig. 11b for the case with distance value. When a host receives the gradient, infrastructural agents spread it

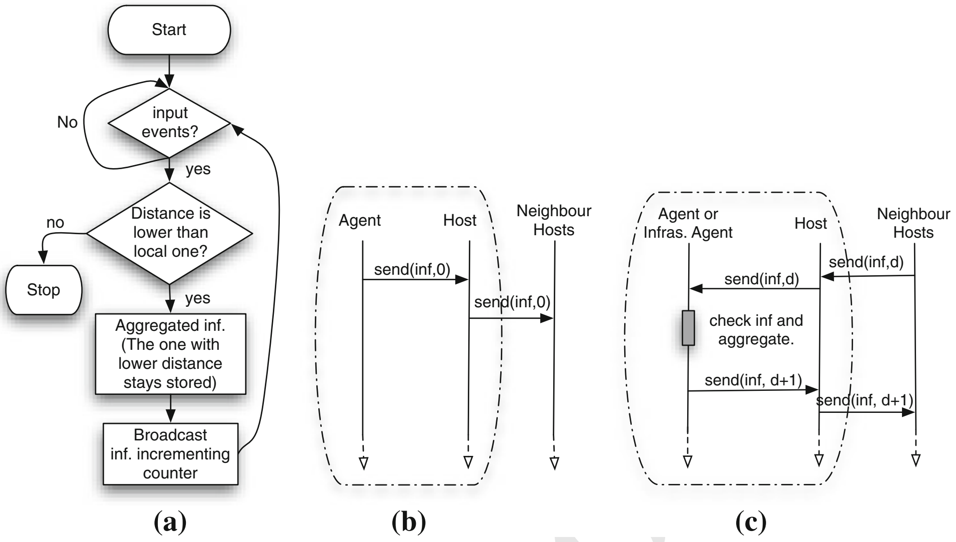


Fig. 11 Gradients: agent behaviour (**a**), initialisation (**b**), agent and infrastructural agent (**c**)

further. Notice that this pattern can be also executed by agents. When an agent receives more than one gradient, it employs aggregation (Aggregation Pattern) as shown on sequence diagram Fig. 11c. For instance, it may filter only the gradient with the lowest distance attribute.

Self-healing gradients (i.e. gradients that adapt to network changes) and their implementations are proposed in (Beal et al. 1969–1975; Viroli et al. 2011).

Known uses the Gradient Pattern has been used in problems such as coordination of swarms of robots (Parunak et al. 2002), coordination of agents in video games (Mamei 2004), or routing in AD-HOC networks (Perkins 1999).

Consequences the Gradient Pattern adds an extra information (distance). Distance can be used to limit the number of hops during the spreading process.

Related Patterns the Gradient Pattern is a composition of the Spreading and Aggregation Patterns, extended with the distance value or concentration information. It is used by the Morphogenesis Pattern (Sect. 5.3.3), the Chemotaxis Pattern (Sect. 5.3.2), and the Quorum Sensing Pattern (Sect. 5.3.4). The Gradient Pattern may be combined with the Evaporation Pattern to create active gradients to support adaptation when agents change theirs positions or network topology changes.

5.2.2 *Digital pheromone pattern*

The Digital Pheromone Pattern is a swarm coordination mechanism based on indirect communication. In this pattern, agents deposit digital pheromones in hosts. A digital pheromone is a mark that spreads a gradient over the environment and persists in the environment for a while, fading away with time. Other agents beyond the communication range can then receive the information conveyed by digital pheromones. Digital pheromones are stored in the hosts and stay active even when agents that deposited digital pheromones disappear. Digital pheromones can be identical to each others, like in Ant Colony Optimisation Algorithm (Dorigo 1999) or can be specialised to a specific task, like in swarming vehicle control (Sauter et al. 2005). Digital pheromones are a particular case of stigmergy communication. Stigmergy is more general and stands for any indirect communication through the environment, not necessarily a sign that behaves like a Digital Pheromone.

Alias none to our knowledge.

Problem coordination of agents in large scale environments using indirect communication.

Solution digital pheromone provides a way to coordinate agent's behaviour using indirect communication in high dynamic environments. Digital pheromones create gradients that spread over the environment, carrying information about their distance and direction. Thus, agents can perceive pheromones from the distance and increase the knowledge about the system. Moreover, as time goes by digital pheromones evaporate, providing adaptation to environmental changes.

Inspiration the Digital Pheromone Pattern takes inspiration from ant colonies. Ant colonies are able to find the shortest paths from the nest to food sources using local interactions and indirect communication based on pheromones. Pheromones are deposited in the environment by ants to mark the path they are following from the nest to the food source and back. Pheromones quickly evaporate so they must be continuously released to maintain the information of the path. Colonies are able to adapt to environment changes (such as, new obstacles, new food sources, food sources that become empty, etc…).

Forces the implementation of the Digital Pheromone Pattern involves the implementation of the Gradient and Evaporation Patterns in order to create an *active gradient*

(Nagpa 2004). The main difference between active gradients and digital pheromones is that pheromone involves indirect communication, while a gradient spreads from agents to agents. Thus, the main forces to consider are the following: (i) as for the Evaporation Pattern, how much and how frequent evaporation is used at each iteration; (ii) as for the Gradient Pattern, the Digital Pheromone Pattern is composed of the Aggregation and Spreading Patterns, thus, the more frequent the spreading of pheromone, the higher the bandwidth used. In addition, spreading pheromones to far away distances, allows more agents to receive the information, but consumes more memory and bandwidth.

Entities-Dynamic-Environment agents are the only entities that can deposit pheromones. Pheromones are deposited in hosts, infrastructural agents then apply spreading, aggregation, and evaporation mechanisms (see Appendix Table 2). Thus, pheromones are spread though the network, aggregated in each host when two or more pheromones' information arrive, and evaporated along the time until they disappear. During a pheromone life time, the pheromone can be perceived even beyond the host's communication range, where the pheromone is actually hosted, due to the effect of the Spreading Pattern.

The transition rule for the Digital Pheromone Pattern is obtained composing the three basic patterns: Spreading, Aggregation and Evaporation, as shown in Transition Rules (8).

$$
\begin{aligned}
&\texttt{spreading} :: \langle L, [PhV, C]\rangle \xrightarrow{r_{spr}} \langle L_k, [PhV - \Delta PhV, C]\rangle\\
&\quad where\ L_k = random(\{L_1, \ldots, L_n\})\\
&\texttt{aggregation} :: \langle L, [PhV_1, C]\rangle, \ldots, \langle L, [PhV_n, C]\rangle \xrightarrow{r_{aggr}} \langle L, [PhV_i, C]\rangle\\
&\quad where\ PhV_i = max(\{PhV_1, \ldots, PhV_n\})\\
&\texttt{evaporation} :: \langle L, [PhV, C]\rangle \xrightarrow{r_{ev}} \langle L, [PhV', C]\rangle\\
&\quad where\ PhV' = PhV * Ev_{\mathrm{factor}}
\end{aligned}
\tag{8}
$$

Similar to the Gradient Pattern, the first transition rule models the spreading of information modifying the *PhV* concentration attribute by decreasing its value by a ΔPhV interval, representing for instance the distance between two locations. The selection of the target location is the same as for the Gradient Pattern. The second transition rule models the corresponding case of aggregation where only the pheromone with the biggest value is kept. The third transition rule models the evaporation of pheromones, with the Ev_{factor} in the range [0..1].

Implementation digital pheromones are usually implemented using multiplicative static evaporation (i.e. the same evaporation factor is used periodically over the pheromone's information). Independently of the patterns used to implement the Digital Pheromone Pattern, pheromones can be deposited in hosts, (i.e. following the proposed model), simulated by software (Sauter et al. 2005), or implemented using RFID sensors (Mamei 2007). In the Digital Pheromone Pattern, the agents just deposit pheromones and sense from them. Infrastructural Agents are in charge of spreading, aggregating and evaporating the pheromones. The way the agents exploit the digital pheromones involves new patterns that are explained in the next sections.

Known uses digital pheromones have been used mainly in autonomous coordination of swarming UAVs (Parunak et al. 2002; Sauter et al. 2005). Moreover, applications of digital pheromones can be found in the Ant Foraging Pattern description (Sect. 5.3.1).

Consequences as reported in (Sauter et al. 2005), the implementation of Digital Pheromones for swarm coordination provides the following issues to the system: (1) simplicity, compared with the logic necessary in a centralised approach, (2) scalability, the digital pheromones work in a totally decentralised manner, i.e. they are applicable in large scale MAS, and (3) robustness, due to decentralisation and the continuous self-organising process the digital pheromones provide, some agents may fail but the system is robust enough to overcome these failures.

Related Patterns the Digital Pheromone Pattern is composed of the Evaporation and the Gradient Patterns, the latter itself composed of the Aggregation and the Spreading Patterns, so that we can say that the Digital Pheromone Pattern involves the basic patterns Spreading and Evaporation. All these patterns are described in Appendix Table 2. The Digital Pheromone Pattern is exploited by the Ant Foraging Pattern (Sect. 5.3.1) from the high level patterns.

5.2.3 Gossip pattern

The goal of the Gossip Pattern is to obtain a shared agreement about the value of some parameters in the system in a decentralised way. All the agents in the system collaborate to progressively reach this agreement: all of them contribute with their knowledge by aggregating their own knowledge with the neighbours' knowledge and by spreading this aggregated knowledge. Thus, the Aggregation Pattern increases the knowledge and reduces the uncertainty of a single agent by taking into account the knowledge of other agents. Gossip was proposed as an Amorphous computing primitive mechanism by Abelson et al. (2000).

Alias none to our knowledge.

Problem in large-scale systems, agents need to reach an agreement, shared among all agents, with only local perception and in a decentralised way.

Solution information spreads to neighbours, where it is aggregated with local information. Aggregates are spread further and their value progressively reaches the agreement.

Inspiration gossip is inspired from the human social behaviour linked to spreading rumors. People add their own information to information received from other people, they increase their knowledge and spread this knowledge further. When the process is repeated several times, people start to share the same knowledge that results from the sharing of the knowledge of different people.

Forces the Gossip Pattern is composed of the Spreading and Aggregation Patterns. It thus presents the same trade-offs (see Sects. 5.1.1, 5.1.2). As in spreading, the main problem of gossip is the network overload that is produced by the continuous broadcast performed by the agents. In order to reduce the network overload, optimised broadcast can be applied (e.g. not all the neighbours receive the information). The number of neighbours that receive the information is the trade-off of this pattern. The more the neighbours that receive the information, the more robust the system is in the case of failures, but more network overload is produced. Robustness is linked with the network density, higher nodes' adjacency leads to a more robust system.

Entities-Dynamics-Environment the entities involved in the gossip mechanism are agents, infrastructural agents and hosts. Gossip is a composed pattern. The dynamics between the entities is then the same as for aggregation and spreading. Analogously to spreading, only an agent can initiate the process. When one agent desires to initiate a gossip process, it sends the information (e.g. parameters and values) to a subset of its neighbours. If an agent is hosted in one of the neighbouring nodes, the agent gets the information, aggregates the information received with its own information and re-sends the aggregated information to a subset of its own neighbours nodes. The same behaviour is produced by the infrastructural agents when no agent is hosted in one host and the host receives an information, in this case the Infrastructural Agent aggregates all the received information and re-sends it. One agent or infrastructural agent ends the gossip process when the information received and the information previously sent are the same, that means that an agreement has been reached.

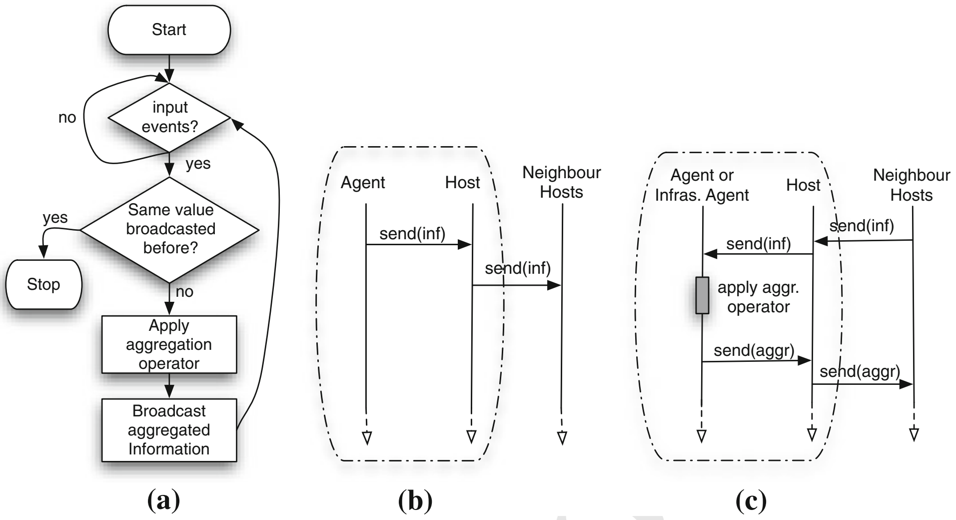
Transition Rules (9) describe gossip. Information received from the neighbours (denoted with the attribute *Recd*) is aggregated to local information and sent to a set of neighbours.

The first transition rule models the spreading of information to a set of locations within the neighbourhood, without modifying its content C, but indicating that the information is sent by a neighbour. As for the spreading, the set of such locations cannot be empty, cannot be composed of L only, but can be composed of all the neighbourhood of L including L itself. The second transition rule models the aggregation of the information received with the local information producing a smallest set of information that the agent then broadcasts again. The process finishes when there is no more broadcast in the system that means, the agents have reached an agreement (i.e. the information received by an agent is the same as its own knowledge).

Implementation regarding implementation, optimised broadcast can be applied. One interesting example of implementation appears in (Haas et al. 2006), where a probabilistic gossip is proposed. It was demonstrated that executing the gossip (broadcast) with a probability between 0.6 and 0.8 is enough to ensure that almost every node gets the message in almost every execution. This optimisation decrements the number of messages by 35 %. Figure 12a shows the flow chart for the standard gossip mechanism where the information spreads using the broadcast. Figure 12b shows the interaction between the agent that initiates the gossip process, the host where the agent is running and the neighbour hosts. Once the gossip has started, the agents and infrastructural agents follow the behaviour presented in Fig. 12c.

Known uses Kempe et al. (2003) analyse a simple gossip-based protocol for the computation of sums, averages, random samples, quantiles, and other aggregate functions. Norman et al. (2010) propose a gossip algorithm where the aggregation is based on Evolutionary Algorithm, and apply this mechanism for coordinating large convention spaces (finding a common vocabulary (lexicon) in their case). The Evolutionary Algorithm approach keeps the diversity throughout the agreement process (not 100 % of agents get the same agreement), this guarantees that when the environment changes the system can quickly achieve a new agreement. It was demonstrated that this approach is resilient to unreliable communications and guarantees the robust emergence of conventions.

$$
\begin{aligned}
&\texttt{spreading} :: \langle L, C\rangle \xrightarrow{r_{spr}} \langle L_1, [Recd, C]\rangle, \ldots, \langle L_n, [Recd, C]\rangle \\
&\quad \mathit{where}\ \{L_1, \ldots, L_n\} = \nu(L) \\
&\texttt{aggregation} :: \langle L, C_1\rangle, \ldots, \langle L, C_m\rangle, \langle L, [Recd, C_{m+1}]\rangle, \ldots, \langle L, [Recd, C_n]\rangle \xrightarrow{r_{aggr}} \langle L, C'_1\rangle, \ldots, \langle L, C'_k\rangle \\
&\quad \mathit{where}\ \{C'_1, \ldots, C'_k\} = \alpha(\{C_1, \ldots, C_n\})
\end{aligned}
\tag{9}
$$



(a) **(b)** **(c)**

Fig. 12 Gossip: agent behaviour (**a**), initialisation (**b**) and interactions with the host and neighbouring hosts (**c**)

Consequences the main advantage of gossip is the robustness. Even in the presence of failures, the pattern is able to reach the agreement. Moreover, gossip provides a continuous adaptation when new values arrive in the system.

Related Patterns the Gossip Pattern is composed of the Spreading Pattern (Sect. 5.1.1) and the Aggregation Pattern (Sect. 5.1.2).

5.3 High-level patterns

This section describes the three high level patterns used in the literature whose contribution in different fields have been demonstrated. For instance, other interesting applications using the Gradient exist in the literature, however their contributions are only focused on one field and no generalisation has been proposed. We present here only those patterns that have been widely accepted and used as mechanisms.

5.3.1 *Ant foraging pattern*

Ant foraging is the activity where a set of ants collaborate to find food. The Ant Foraging Pattern is a decentralised collaborative search pattern. Mainly, the Ant Foraging Pattern has been applied to optimisation problems and used for swarm robotics.

Aliases Ant Colony Optimisation (Dorigo 2002).

Problem large scale optimisation problems that can be transformed into the problem of finding the shortest path on a weighted graph.

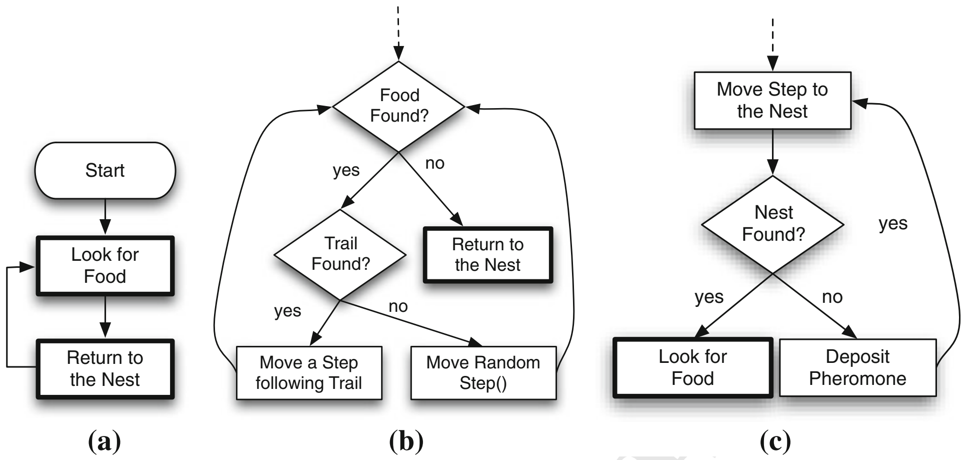
Solution the Ant Foraging Pattern provides rules to explore the environment in a decentralised manner and to exploit resources.

Inspiration the Ant Foraging Pattern is inspired by the Ant Colony Foraging behaviour. In ant colonies, ants coordinate their behaviour to find the shortest path from the nest to the food. Ant colonies use a stigmergic communication means, i.e. ants modify the environment by depositing a chemical substance called pheromone. This pheromone drives the behaviour of other ants in the colony, pheromone concentrations being used to recruit other ants. Following the highest pheromone concentration, ants find the shortest path from the nest to the food, and adapt this path when obstacles appear or when food is depleted.

Forces each ant has a probability of following the gradient produced by the pheromones. When one ant is not following the gradient, it walks randomly in the environment looking for new resources (exploration). When the probability of exploration is high (i.e. low probability of following the gradient), ants adapt faster to environmental changes but are slower in reaching the resources (exploitation). Whereas, with a low exploration (i.e. high probability of following the gradient), ants are quick in exploiting the resources since most of the ants follow the path to the resource. However, due to the lack of exploration, when the resource is depleted the ants spend more time to find new resources and adaptation is slower. Additionally, the Ant Foraging Pattern presents the same forces as the Digital Pheromone Pattern (Sect. 5.2.2). If the evaporation rate of the pheromone is too low, the pheromone scent does not evaporate quickly enough and stays where it has been laid down. The environment gets filled with pheromone and the exploitation is not efficient. A high evaporation rate causes the pheromone to evaporate before ants can build a path and maintain it, reducing the exploitation and incrementing the exploration.

Entities-Dynamic-Environment the entities involved in the Ant Foraging Pattern are the same as for the Digital Pheromone Pattern (Sect. 5.2.2). When one agent senses the presence of a digital pheromone, it decides to follow the gradient or to move randomly.

Fig. 13 Ant foraging: general flow (**a**), looking for food (**b**), returning to the nest (**c**)



Transition Rule (10) describes the ant foraging behaviour. It extends Transition Rule (8) that creates the field of pheromones.

$$
\begin{aligned}
&\texttt{up\_move} :: \langle L,[PhV_1,C]\rangle,\ldots,\langle L_n,[PhV_n,C]\rangle \overset{r_{umove}}{\longrightarrow} \langle L_i,[PhV_i,C]\rangle\\
&\quad \mathit{where}\ PhV_i = \mathit{max}(\{PhV_1,\ldots,PhV_n\})\\
&\texttt{random\_move} :: \langle L,C]\rangle \overset{r_{rmove}}{\longrightarrow} \langle L_i,C]\rangle\\
&\quad \mathit{where}\ L_i = \mathit{random}(\{L_1,\ldots,L_n\})
\end{aligned} \tag{10}
$$

The first rule models an agent that senses the values of the pheromone field in its location and in the neighbourhood, and then follows the direction of the highest gradient value to find food. The second rule models an agent that moves randomly. Both rules are subject to a rate which regulates the exploitation vs exploration activities.

Implementation according to some exploration probability, agents either follow scouts (i.e. are recruited to exploit food), or perform some random search. In the case of ants, scouts deposit pheromones in their environment, that are later sensed by other ants to find food sources. Figure 13a shows the general behaviour of ants, Fig. 13b shows the behaviour of ants looking for food, following a trail or taking a random path, finally Fig. 13c show the return to the nest, dropping pheromone, once a piece of food has been found.

Known uses the Ant Foraging Pattern has been mainly applied in Ant Colony Optimisation (ACO) (Dorig 1992) in applications such as, scheduling (Blu 2005; Martens et al. 2007), vehicle routing problems (Bachem 1996; Secomand 2000; Toth 2002), or assignment problems (Lourenço 1998).

Consequences the system achieves high quality performance in NP-Hard search problems.

Related Patterns the Ant Foraging Pattern exploits the Digital Pheromone Pattern (Sect. 5.2.2). Thus, the Ant Foraging Pattern uses Evaporation, Spreading and Aggregation Patterns (see Appendix Table 2 for details about these patterns).

5.3.2 Chemotaxis pattern

The Chemotaxis Pattern provides a mechanism to perform motion coordination in large scale systems. Chemotaxis was initially proposed by Nagpal (Nagpa 2004). The Chemotaxis Pattern extends the Gradient Pattern: agents identify the gradient direction to decide the direction of their next movements.

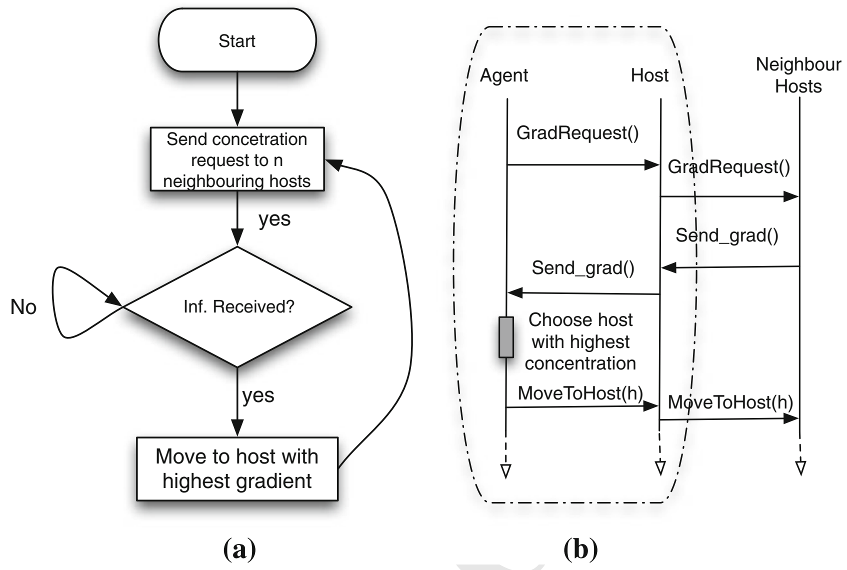
Alias none to our knowledge.

Problem decentralised motion coordination aiming at detecting sources or boundaries of events.

Solution agents locally sense gradient information and follow the gradient in a specified direction (i.e. follow higher gradient values, lower gradient values, or equipotential lines of gradients).

Inspiration in biology, chemotaxis is the phenomenon in which single or multi-cellular organisms direct their

Fig. 14 Chemotaxis: agent behaviour (**a**), agent interaction (**b**)



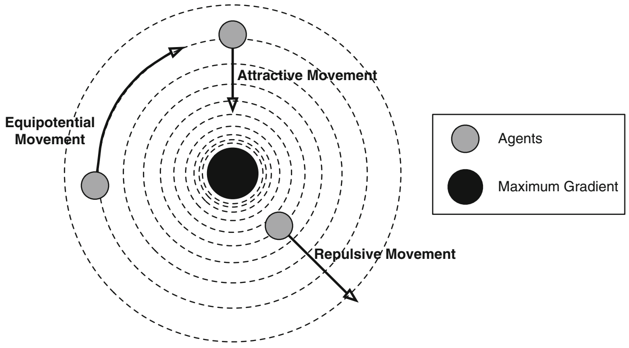


Fig. 15 Chemotaxis pattern—adapted from (De Wolf 2007)

movements according to certain chemicals present in their environment. Examples in nature include: leukocyte cells moving towards a region of a bacterial inflammation or bacteria migrating towards higher concentrations of nutrients (Wolpert et al. 2007). Notice that in biology, chemotaxis is also a basic mechanism of morphogenesis. It guides cells during development so that they will be placed in the final right position. In this paper, following (Nagpa 2004), the term chemotaxis is used as motion coordination following gradients, while the term morphogenesis is used for triggering specific behaviours based on relative positions determined through a gradient.

Forces the Chemotaxis Pattern exploits the Gradient Pattern (see Sect. 5.2.1 to find information about the forces involved in the Gradient Pattern). In the Chemotaxis Pattern the communication range plays an important role. When the communication range is long, agents move faster following the gradients. This, however, causes problems for precisely locating sources. On the other hand, short communication ranges need a higher number of hops to follow the gradient, but they allow to find sources with high precision.

Entities-Dynamic-Environment the concentration of gradient guides the agents' movements in three different ways, as shown in Fig. 15: (1) attractive movement, when agents change their positions following higher gradient values, (2) repulsive movement, when agents follow lower gradient values, incrementing the distance between the agent and the gradient source, and (3) equipotential movement, when agents follow gradients between thresholds.

Given the Transition Rule (7) that creates the gradient, Transition Rule (11) determines the agent movement towards the highest, lowest, or equipotential gradient value (depending on the cases).

$$\texttt{move} :: \langle L, [D_1, C]\rangle, \ldots, \langle L_n, [D_n, C]\rangle \xrightarrow{r_{move}} \langle L_i, [D_i, C]\rangle$$

$$where\ D_i = min/max/equal(\{D_1, \ldots, D_n\}) \qquad (11)$$

Implementation chemotaxis can be implemented in two different ways. First, using gradients existing in the environment to coordinate the agent's positions or directions (e.g. using attractive and equipotential movements to detect the contour of diffuse events (Ruairí 2007), or using attractive movements to detect diffuse event sources (Fernandez-Marquez et al. 2012) through a multi-agent approach over a sensor network infrastructure). Second, using gradient fields generated by agents (e.g. using a gradient-based approach to coordinate the position of bots in the Quake 3 Arena video game (Mamei 2004)). Diagram 14a, b show a particular case of implementation, where agents get information about neighbouring gradients, before taking a decision about where to go next. As shown in Diagram 14a, each agent chooses *n* random neighbouring host and sends them a gradient concentration request. The agent chooses the neighbouring host that has a highest gradient concentration and moves there. By repeating this process the agent is able to find the gradient source.

Known uses Mamei et al. (2004) use Chemotaxis to coordinate the position of a swarm of simple mobile robots. Chemotaxis is also used in (Viroli et al. 2011), where chemotaxis is applied to route messages in pervasive computing scenarios.

Related Patterns the Chemotaxis Pattern extends the Gradient Pattern (Sect. 5.2.1).

5.3.3 Morphogenesis pattern

The goal of the Morphogenesis Pattern is to select different agent's behaviour depending on the agent's position in the system. The Morphogenesis Pattern exploits the Gradient Pattern: relative spatial position information is assessed through one or multiple gradient sources generated by other agents. Morphogenesis was proposed as a self-organising mechanism in (Mamei et al. 2006; Sudeikat 2008). The morphogenesis process in biology has been considered as an inspiration source for gradient fields.

Alias none to our knowledge.

Problem in large-scale decentralised systems, agents decide on their roles or plan their activities based on their spatial position.

Solution specific agents spread morphogenetic gradients. Agents assess their positions in the system by computing their relative distance to the morphogenetic gradients sources.

Inspiration in the biological morphogenetic process some cells create and modify molecules (through aggregation) which diffuse (through spreading), creating gradients of molecules. The spatial organisation of such gradients is the morphogenesis gradient, which is used by the cells to differentiate the role that they play inside the body, e.g. in order to produce cell differentiations.

Forces the forces presented in this pattern are the same as the ones of the Gradient Pattern (Sect. 5.2.1).

Entities-Dynamic-Environment the entities involved in the morphogenesis process are Agents, Hosts, and Infrastructural Agents. At the beginning, some of the agents spread one or more morphogenesis gradients, implemented using the Gradient Pattern. Other agents sense the morphogenetic gradient in order to calculate their relative positions. Depending on their relative positions, the agents adopt different roles and coordinate their activities in order to achieve collaborative goals.

Given Transition Rule (7) that creates the gradient, Transition Rule (12) models an agent sensing its local gradient values and adapting its behaviour depending on its relative position with respect to the gradient source.

$$\texttt{state\_evolution} :: \langle L, [D, State, C]\rangle \xrightarrow{r_{move}} \langle L, [D, State', C]\rangle$$
$$\text{where } State' = \pi(D) \qquad (12)$$

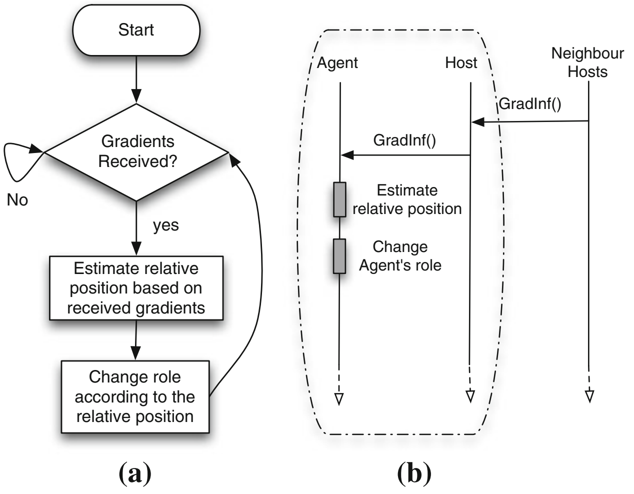


Fig. 16 Morphogenesis: agent behaviour (**a**), agent interaction (**b**)

Function $\pi(D)$ changes the state variables of the agent, evolving its state according to the information it locally perceives in the environment.

Implementation an interesting implementation of the morphogenesis gradient to estimate positions is proposed in (Bea 2009), where a self-healing gradient algorithm with a tunable trade-off between precision and communication cost is proposed. In (Mamei et al. 2004) the motion coordination of a swarm of robots is implemented by using both Morphogenesis and Chemotaxis Patterns (Sect. 5.3.2). Diagram 16a, b show agents estimating their position in response to gradient information propagated by neighbouring hosts.

Known uses the Morphogenesis Pattern is used to implement control techniques for modular self-reconfigurable robots (meta-morphic robots) (Bojinov et al. 2001) . Morphogenesis is also employed to create a robust process for shape formation on a sheet of identically programmed agents (origami) (Nagpa 2002).

Consequences the Morphogenesis Pattern equips the agents with a mechanism to coordinate their activities based on their relative positions. Like the other mechanisms previously presented, robustness and scalability are properties ensured by this pattern.

Related Patterns the Morphogenesis Pattern extends the Gradient Pattern (Sect. 5.2.1). The Morphogenesis Pattern can be combined with the Digital Pheromone Pattern where the role and behaviour of the agents depend on the distances to the pheromone sources.

5.3.4 Quorum sensing pattern

Quorum sensing is a decision-making process for coordinating behaviour and for taking collective decisions in a decentralised way. The goal of the Quorum Sensing Pattern is to provide an estimation of the number of agents (or of the density of the agents) in the system using only local interactions. The number of agents in the system is crucial in those applications, where a minimum number of agents are needed to collaborate on specified tasks.

Alias none to our knowledge.

Problem collective decisions in large-scale decentralised systems, requiring a threshold number of agents or estimation of the density of agents in a system, using only local interactions.

Solution the Quorum Sensing Pattern allows to take collective decisions through an estimation by individual agents of the agents' density (assessing the number of other agents they interact with) and by determination of a threshold number of agents necessary to take the decision.

Inspiration the Quorum Sensing Pattern is inspired by the Quorum Sensing process (QS), which is a type of intercellular signal used by bacteria to monitor cell density for a variety of purposes. An example is the bioluminescent bacteria (Vibrio Fischeri) found in some species of squids. These bacteria self-organise their behaviour to produce light only when the density of bacteria is sufficiently high (Miller 2001). The bacteria constantly produce and secrete certain signalling molecules called auto-inducers. In presence of a high number of bacteria, the level of auto-inducers increases exponentially (the higher the auto-inducer level a bacteria detects, the more auto-inducer it produces). Another interesting example is given by the colonies of ants (Leptothorax albipennis) (Sahin 2002), when the colony must find a new nest site. A small portion of the ants search for new potential nest sites and assess their quality. When they return to the old nest, they wait for a certain period of time before recruiting other ants (higher assessments produce lower waiting periods). Recruited ants visit the potential nest site and make their own assessment about the nest quality returning to the old nest and repeating the recruitment process. Because of the waiting periods, the number of ants present in the best nest will tend to increase. When the ants in this nest sense that the rate at which they encounter other ants exceeds a particular threshold, the quorum number is reached. Other swarms like honeybees or wasps use the same technique for nest finding.

Forces the Quorum Sensing Pattern uses gradients presenting the same parameters as the Gradient Pattern (Sect. 5.2.1). The threshold, indicating that the quorum number has been reached, triggers the collaborative behaviour. Quorum Sensing provides an estimation of the density of agents in the system. However, this pattern does not provide a solution to calculate the number of agents necessary to carry out a collaborative task (i.e. to identify the threshold value).

Entities-Dynamic-Environment the entities involved in the Quorum Sensing Pattern are the same as in the Gradient Pattern. Namely, Agents, Hosts, and Infrastructural Agents. The concentration is estimated by the aggregation of the gradients.

The transition rule for the Quorum Pattern can be modelled through Transition Rule (12), where the evolution function $\pi(D)$ has the form given by Eq. (13):

$$\pi(D) = \begin{cases} State & \text{if } D \leq threshold \\ State' & \text{if } D > threshold \end{cases} \tag{13}$$

Implementation there is no specific implementation for the Quorum Sensing Pattern. However, biological systems presented above give us some ideas about how to implement the pattern. Here we propose two different approaches to implement the Quorum Sensing Pattern: (1) to use the Gradient Pattern to simulate the auto-inducers like in the bioluminescent bacteria. In this case the gradient concentration provides the agents with an estimation of the agents' density; (2) as in ants' systems, the agents' density can be estimated through the frequency to which agents are in communication range. The use of gradients provides better estimations than the use of frequencies. However, it is more expensive computationally and it requires more network communications. Diagram 17a, b show agents identifying whether the concentration gradient has reached the threshold, in response to gradient information propagated by neighbouring hosts.

Known uses the Quorum Sensing Pattern is used to increase the power saving in Wireless Sensor Networks (Britton 2004). Quorum sensing permits to create clusters based on the structure of the observed parameters of interest, and then only one node for each cluster sends the

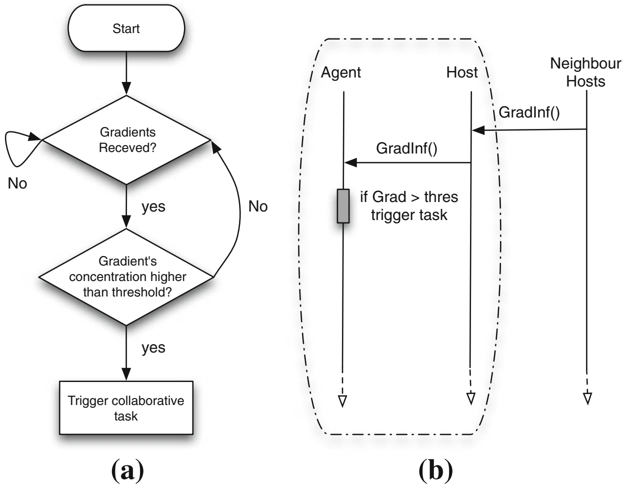


Fig. 17 Quorum sensing: agent behaviour (**a**), agent interaction (**b**)

information on behalf of the quorum. Another known example is the coordination of Autonomous Swarm Robots (Sahin 2002).

Consequences each agent can estimate the density of nodes or the density of other agents in the system using only local information received from neighbours, even when the system is really large and agents are anonymous.

Related Patterns the Quorum Sensing Pattern, depending on its implementation, uses the Gradient Pattern (Sect. 5.2.1).

5.3.5 Flocking pattern

Flocking is a kind of self-organising motion coordination behaviour of a herd of animals of similar size and body orientation, often moving en masse or migrating in the same direction and with a common group objective. The Flocking Pattern is able to control dynamic pattern formations and move the agents over the environment while keeping the formation pattern, interconnections between them and avoiding collisions.

Different disciplines have been interested in the emergent behaviour of flocking, swarming, schooling and herding. Several examples can be found in (Olfati-Sabe 2006). The forces that drive the flocking behaviour were proposed in 1986 by Craig W. Reynolds (Reynold 1987). They are known as Reynolds rules: (1) cohesion (flock centering), (2) separation (obstacle avoidance and crowd avoidance) and (3) alignment (velocity and direction matching). Cohesion captures the intuition that individuals try to keep close to nearby flockmates because they always try to move towards the flocking center. Separation pursues collision avoidance with nearby flockmates and other obstacles. Alignment is related to the ability to move the flock with all the individuals at the same speed. Flocking is typically used for motion coordination of large scale MAS, mainly 2D or 3D simulations.

Problem dynamic motion coordination and pattern formation of swarms.

Solution the Flocking Pattern provides a set of rules for moving groups of agents over the environment while keeping the formation and interconnections between them.

Inspiration this pattern is inspired by the behaviour of a group of birds when they are foraging or flying and by schools of fish when they are avoiding a predator attack or foraging. For example, when a school of fish is under a predator attack, the movement of the first fish sensing the predator presence, produces a fast movement alerting the other fishes by waves of pressure sent through the water. The schools of fish then changes its formation for avoiding the predator attack, recovering the initial formation after the attack. It is similar for obstacle avoidance.

Forces parameters such as, avoidance distance, maximum velocity and maximum acceleration must be tuned to achieve the desired motion coordination.

Entities-dynamic-environment the entities participating in the Flocking Pattern are only Agents using direct communication. Basically, agents sense the position of their neighbours and keep a constant desired distance. When the distance changes due to external perturbations, each agent responds in a decentralised way to control the distance and to recover the original formation pattern.

The transition rule for the Flocking Pattern is formalised in Transition Rule (4), where the specific instance of ρ for computing the new position is described in the following.

Implementation details about the algorithm and theory can be found in (Olfati-Sabe 2006). Here we present some basic concepts about the algorithm and the implementation. Analogously to the free-flocking algorithm presented in (Olfati-Sabe 2006), each agent's motion is controlled by Eq. (14).

$$\mathbf{u}_i = f_i^g + f_i^d + f_i^\gamma \qquad (14)$$

where f_i^g is a gradient based term that represents the cohesion and separation Reynolds rules (1) & (2). f_i^d is a velocity consensus/alignment term that represents the alignment rule (3). Finally, f_i^γ is the navigational feedback term that drives the group to the objective.

Figures 18 represents two agents that coordinate their behaviour according to the first term (cohesion and separation): (*a*) agents are attracted to each other, because they

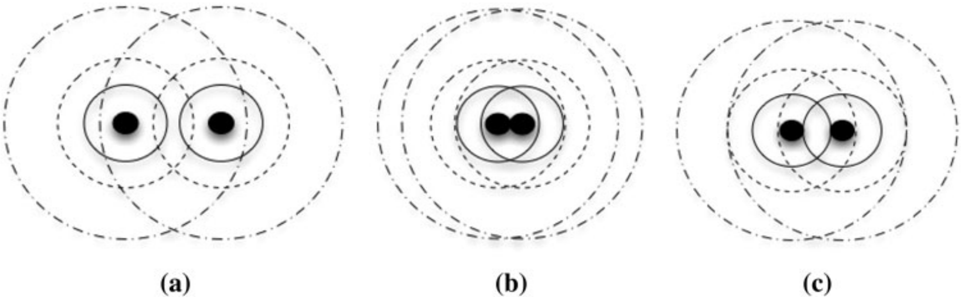


Fig. 18 Metric distance model—movements

are situated in an attracting zone; (*b*) agents repel each other because they are too close; finally, in (*c*) agents are in the neutral zone where the term becomes zero. When all the agents of the flocks are situated in the neutral area, they form a stress-free structure. Analogously to the Repulsion Pattern (Sect. 5.1.4), the interactions between the entities participating in the Flocking Pattern are the same as the interactions shown in the Repulsion Pattern (Sect. 5.1.4). The only difference is that the Flocking Pattern applies more rules, not only repulsion.

Known uses the first application of the Flocking Pattern was modelling animal behaviour for movies. Specifically, it was used to generate realistic crowds moves. Flocking has also been used to control the behaviour of Unmanned Air Vehicles (UAVs) (Crowther 2002), Autonomous mobile robots (Hayes 2002; Jadbabaie et al. 2003), Micro or Miniature Aerial Vehicles (MAV) (Nardi et al. 2006) and Mobile Sensor Networks (La 2009, 2009).

Consequences flocking tries to generalise the behaviour of flocking, independently of individuals (birds, penguins, fish, etc.). Its behaviour does not depend on the methods used for the generation of agents' trajectories. The Flocking Pattern provides robustness and self-healing properties when faced with agents' failures and communication problems.

Related Patterns the Flocking Pattern extends the Repulsion Pattern (Sect. 5.1.4). In fact, repulsion can be seen as a simplification of the Flocking Pattern where only the repulsion vector is taken into account for calculating the next position.

6 Conclusion and future work

This paper proposes a catalogue of bio-inspired self-organising mechanisms uniformly expressed as modular and reusable design patterns, which we organised into different layers. On the one hand the design pattern description allows us to give a detailed information about how and when each mechanisms should be used. On the other hand, the classification and relations between the mechanisms provide a better understanding of their behaviours, and allows engineers to design and implement bio-inspired systems by adding modular bio-inspired functionalities. Future work will consider the inclusion of additional mechanisms in the catalogue, further investigation of the patterns' usage and how applications can be built on top of a bio-inspired framework where the different mechanisms can be provided by the underlying environment and requested on demand (preliminary results can be found in (Fernandez-Marquez et al. 2011)), thus, allowing applications to be designed and implemented in a modular way (i.e. reusing code).

Appendix

1. Design patterns summary

Table 2 summarises each design pattern giving the problem its solves and the solution it provides.

Table 2 Patterns table

Pattern's name	Problem and solution
Spreading (Sect. 5.1.1)	In systems, where agents perform only local interactions, agents' reasoning suffers from the lack of knowledge about the global system. a copy of the information (received or held by an agent) is sent to neighbours and propagated over the network from one node to another. Information spreads progressively over the system and reduces the lack of knowledge of the agents while keeping the constraint of the local interaction
Aggregation (Sect. 5.1.2)	In large systems, excess of information produced by the agents may produce network and memory overloads. Information must be distributively processed in order to reduce the amount of information and to obtain meaningful information. aggregation consists in locally applying a fusion operator to process the information and synthesise macro information. This fusion operator can take many forms, such as filtering, merging, aggregating, or transforming (Chen 2002)
Evaporation (Sect. 5.1.3)	Outdated information cannot be detected and it needs to be removed, or its detection involves a cost that needs to be avoided. Agent decisions rely on the freshness of the information presented in the system, enabling correct responses to dynamic environments.evaporation is a mechanism that periodically reduces the relevance of information. Thus, recent information becomes more relevant than older information
Repulsion (Sect. 5.1.4)	Agents' movements have to be coordinated in a decentralised manner in order to achieve a uniform distribution and to avoid collisions among them. The Repulsion Pattern creates a repulsion vector that guides agents to move from regions with high concentrations of agents to regions with lower concentrations. Thus, after few iterations agents reach a more uniform distribution in the environment
Gradients (Sect. 5.2.1)	Agents belonging to large systems suffer from lack of global knowledge to estimate the consequences of their actions or the actions performed by other agents beyond their communication range. Information spreads from the location it is initially deposited and aggregates when it meets other information. During spreading, additional information about the sender's distance and direction is provided: either through a distance value (incremented or decremented); or by modifying the information to represent its concentration (lower concentration when information is further away). Thus, agents that receive gradients have information that come from beyond their communication range, increasing the knowledge of the global system not only with gradients information but also with the direction and distance of the information source. During the aggregation process, a filter operator keeps only the information with the highest (or lowest) distance, or it modifies the concentration. Gradients can deal with network topology changes. In this case the information spreads periodically and is subject to evaporation, reducing its relevance along the time, and enabling the gradients to adapt to networks topology changes. Such gradients are called active gradients (Clement 2003)

Table 2 continued

Pattern's name	Problem and solution
Digital pheromone (Sect. 5.2.2)	Coordination of agents in large scale environments using indirect communication. Digital pheromone provides a way to coordinate agent's behaviour using indirect communication in high dynamic environments. Digital pheromones create gradients that spread over the environment, carrying information about their distance and direction. Thus, agents can perceive pheromones from the distance and increase the knowledge about the system. Moreover, as time goes by digital pheromones evaporate, providing adaptation to environmental changes
Gossip (Sect. 5.2.3)	in large-scale systems, agents need to reach an agreement, shared among all agents, with only local perception and in a decentralised way. Information spreads to neighbours, where it is aggregated with local information. Aggregates are spread further and their value progressively reaches the agreement
Ant foraging (Sect. 5.3.1)	Large scale optimisation problems that can be transformed into the problem of finding the shortest path on a weighted graph. The Ant Foraging Pattern provides rules to explore the environment in a decentralised manner and to exploit resources
Chemotaxis (Sect. 5.3.2)	Decentralised motion coordination aiming at detecting sources or boundaries of events. agents locally sense gradient information and follow the gradient in a specified direction (i.e. follow higher gradient values, lower gradient values, or equipotential lines of gradients)
Morphogenesis (Sect. 5.3.3)	In large-scale decentralised systems, agents decide on their roles or plan their activities based on their spatial position. specific agents spread morphogenetic gradients. Agents assess their positions in the system by computing their relative distance to the morphogenetic gradients sources
Quorum sensing (Sect. 5.3.4)	Collective decisions in large-scale decentralised systems, requiring a threshold number of agents or estimation of the density of agents in a system, using only local interactions. The Quorum Sensing Pattern allows to take collective decisions through an estimation by individual agents of the agents' density (assessing the number of other agents they interact with) and by determination of a threshold number of agents necessary to take the decision
Flocking (Sect. 5.3.5)	Dynamic motion coordination and pattern formation of swarms. The Flocking Pattern provides a set of rules for moving groups of agents over the environment while keeping the formation and interconnections between them

References

Abelson H, Allen D, Coore D, Hanson C, Homsy G, Thomas F, Knight J, Nagpal R, Rauch E, Sussman GJ, Weiss R (2000) Amorphous computing. Commun ACM 43(5):74–82

Bachem A, HochstŁttler W, Malich M (1996) The simulated trading heuristic for solving vehicle routing problems. Tech. Rep. Discr Appl Math 65:47–72

Beal J (2009) Flexible self-healing gradients. In: SAC '09: proceedings of the 2009 ACM symposium on applied computing. ACM, pp 1197–1201

Beal J, Bachrach J, Vickery D, Tobenkin M (2008) Fast self-healing gradients. In: SAC '08: proceedings of the 2008 ACM symposium on applied computing. ACM, New York, pp 1969–1975

Birman KP, Hayden M, Ozkasap O, Xiao Z, Budiu M, Minsky Y (1999) Bimodal multicast. ACM Trans Comput Syst 17:41–88

Blum C (2005) Beam-aco: hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591

Bojinov H, Casal A, Hogg T (2001) Multiagent control of self-reconfigurable robots

Britton M, Sack L (2004) The secoas project: development of a self-organising wireless sensor network for environmental monitoring. In: The 2nd international workshop on sensor and actor network protocols and applications. Boston

Chen G, Kotz D (2002) Context aggregation and dissemination in ubiquitous computing systems. In: Proceedings of the fourth IEEE workshop on mobile computing systems and applications, WMCSA '02. IEEE Computer Society, Washington, DC, p 105

Cheng J, Cheng W, Nagpal R (2005) Robust and self-repairing formation control for swarms of mobile agents. In: Proceedings of the twentieth national conference on artificial intelligence. AAAI Press, London, pp 59–64

Clement L, Nagpal R (2003) Self-assembly and self-repairing topologies. In: Workshop on adaptability in multi-agent systems, first RoboCup Australian open. AORC

Crowther WJ, Riviere X (2002) Flocking of autonomous unmanned air vehicles. In: The 17th Bristol UAV conference

de Castro LN (2006) Fundamentals of natural computing: basic concepts, algorithms, and applications (Chapman & Hall/CRC computer and information sciences). Chapman & Hall/CRC, Boca Raton

De Wolf T, Holvoet T (2007) Design patterns for decentralised coordination in self-organising emergent systems. Eng Self-Org Syst 4335:28–49

Deneubourg J, Pasteels J, Verhaeghe J (1983) Probabilistic behaviour in ants: a strategy of errors?. J Theor Biol 105(2):259–271

Di Marzo Serugendo G, Gleizes MP, Karageorgos A (eds) (2011) Self-organising software—from natural to artificial adaptation, 1st edn. Natural computing series. Springer, New York

Dorigo M (1992) Optimization, learning and natural algorithms. Ph.D. thesis, Politecnico di Milano, Italy

Dorigo M, Di Caro G (1999) The ant colony optimization meta-heuristic. In: New ideas in optimization. McGraw-Hill, London, pp 11–32

Dorigo M, Sttzle T (2002) The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Handbook of meta-heuristics. Kluwer, Norwell, pp 251–285

Dressler F, Akan OB (2010) A survey on bio-inspired networking. Computer Netw 54(6):881–900

Fernandez-Marquez JL, Arcos JL (2009) An evaporation mechanism for dynamic and noisy multimodal optimization. In: The 11th annual conference on genetic and evolutionary computation, GECCO '09. ACM, pp 17–24

Fernandez-Marquez JL, Arcos JL (2010) Adapting particle swarm optimization in dynamic and noisy environments. In: Proceedings of IEEE congress on evolutionary computation, pp 765–772

Fernandez-Marquez JL, DiMarzo Serugendo G, Arcos JL (2011) Infrastructureless spatial storage algorithms. ACM Trans Auton Adapt Syst 6:15–11526

Fernandez-Marquez JL, Di Marzo Serugendo G, Montagna S (2011) Bio-core: bio-inspired self-organising mechanisms core. In: 6th international ICST conference on bio-inspired models of network, information, and computing systems. LNCS. York

Fernandez-Marquez JL, Lluis AJ, Di Marzo Serugendo G (2012) A decentralized approach for detecting dynamically changing diffuse event sources in noisy WSN environments. Applied artificial intelligence. Taylor & Francis, Bristol (to appear)

Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading

Gardelli L, Viroli M, Omicini A (2007) Design patterns for self-organizing multiagent systems. In: Proceedings of EEDAS

Grégoire E, Konieczny S (2006) Logic-based approaches to information fusion. Inf Fusion 7(1):4–18

Haas ZJ, Halpern JY, Li L (2006) Gossip-based ad hoc routing. IEEE/ACM Trans Netw 14(3):479–491

Hayes AT, Dormiani-tabatabaei P (2002) Self-organized flocking with agent failure: off-line optimization and demonstration with real robots. In: ICRA'02: proceedings of the 2002 IEEE international conference on robotics and automation, pp 3900–3905

Huebel N, Hirche S, Gusrialdi A, Hatanaka T, Fujita M, Sawodny O (2008) Coverage control with information decay in dynamic environments. In: Proceedings of 17th IFAC world congress. Seoul, pp 4180–4185

Jadbabaie A, Lin J, Morse AS (2003) Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans Autom Control 48(6):988–1001

Kempe D, Dobra A, Gehrke J (2003) Gossip-based computation of aggregate information. Foundations of computer science, 2003. In: Proceedings. 44th annual IEEE aymposium on, pp 482–491

Khelil A, Becker C, Tian J, Rothermel K (2002) An epidemic model for information diffusion in MANETs. In: MSWiM '02: proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems. ACM, pp. 54–60

La HM, Sheng W (2009) Flocking control of a mobile sensor network to track and observe a moving target. In: ICRA'09: proceedings of the 2009 IEEE international conference on robotics and automation. IEEE Press, Piscataway, pp 3586–3591

La HM, Sheng W (2009) Moving targets tracking and observing in a distributed mobile sensor network. In: ACC'09: proceedings of the 2009 conference on American control conference. IEEE Press, Piscataway, pp 3319–3324

Lee S, Chung TC (2004) Data aggregation for wireless sensor networks using self-organizing map. In: AIS, pp 508–517

Lourenço HR, Serra D (1998) Adaptive approach heuristics for the generalized assignment problem. Economic working papers series no. 304, Universitat Pompeu Fabra, Department of Economics and Management

Mamei M, Menezes R, Tolksdorf R, Zambonelli F (2006) Case studies for self- organization in computer science. J Syst Archit 52:433–460

Mamei M, Vasirani M, Zambonelli F (2004) Experiments of morphogenesis in swarms of simple mobile robots. J Appl Artif Intell 18:903–919

Mamei M, Zambonelli F (2004) Field-based motion coordination in quake 3 arena. In: Proceedings of the third international joint conference on autonomous agents and multiagent systems, AAMAS '04, vol 3. IEEE Computer Society, pp 1532–1533

Mamei M, Zambonelli F (2007) Pervasive pheromone-based interaction with rfid tags. ACM Trans Auton Adapt Syst 2

Martens D, De Backer M, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. IEEE Trans Evol Comput 11:651–665

Miller MB, Bassler BL (2001) Quorum sensing in bacteria. Annu Rev Microbiol 55(1):165–199

Nagpal R (2002) Programmable self-assembly using biologically-inspired multiagent control. In: 1st intl joint conf. on autonomous agents and multiagent systems: part 1, pp 418–425

Nagpal R (2004) A catalog of biologically-inspired primitives for engineering self-organization. In: Engineering self-organising systems, nature-inspired approaches to software engineering. Springer, New York, pp 53–62

Nardi RD, Holl O, Woods J, Clark, A (2006) Swarmav: a swarm of miniature aerial vehicles. In: The 21st Bristol international UAV aystems conference

Niu R, Varshney PK (2005) Distributed detection and fusion in a large wireless sensor network of random size. EURASIP J Wirel Commun Netw 462–472

Olfati-Saber R (2006) Flocking for multi-agent dynamic systems: algorithms and theory. IEEE Trans Autom Control 51:401–420

Parunak HVD, Purcell M, O'Connell R (2002) Digital pheromones for autonomous coordination of swarming uavs. In: The first AIAA unmanned aerospace vehivales, systems, technologies, and operations, pp 1–9

Perkins CE, Royer EM (1999) Ad-hoc on-demand distance vector routing. In: Proceedings of the second IEEE workshop on mobile computer systems and applications, WMCSA '99. IEEE-CS

Pigozzi G, Hartmann S (2007) Aggregation in multi-agent systems and the problem of truth-tracking. In: The 6th international joint conference on autonomous agents and multiagent systems (AAMAS 07), pp 674 – 676

Puviani M, Di Marzo Serugendo G, Frei R, Cabri G (2012) A method fragments approach to methodologies for engineering self-organising systems. In: ACM transactions on autonomous adaptive systems (to appear)

Ranganathan A, Al-Muhtadi J, Chetan S, Campbell R, Mickunas MD (2004) Middlewhere: a middleware for location awareness in ubiquitous computing applications. In: Proceedings of middleware '04, pp 397–416

Reynolds CW (1987) Flocks, herds, and schools: a distributed behavioral model. In: SIGGRAPH '87: proceedings of the 14th annual conference on computer graphics and interactive techniques. ACM, New York, pp 25–34

Ruairí RM, Keane MT (2007) An energy-efficient, multi-agent sensor network for detecting diffuse events. In: IJCAI'07: proceedings of the 20th international joint conference on artifical intelligence. Morgan Kaufmann Publishers Inc, pp 1390–1395

Sabbineni H, Chakrabarty K (2005) Location-aided flooding: an energy-efficient data dissemination protocol for wireless sensor networks. IEEE Trans Comput 54:36–46

Sahin E, Franks NR (2002) Measurement of space: from ants to robots. In: WGW 2002: EPSRC/BBSRC international workshop biologically-inspired robotics

Salazar N, Rodriguez-Aguilar JA, Arcos JL (2010) Robust coordination in large convention spaces. AI Commun 23(4):357–372

Sauter JA, Matthews R, Van Dyke Parunak H, Brueckner SA (2005) Performance of digital pheromones for swarming vehicle control. In: Proceedings of the fourth international joint conference on autonomous agents and multiagent systems, AAMAS '05. ACM, pp 903–910

Secomandi N (2000) Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. Comput Oper Res 27(11–12):1201–1225

Sudeikat J, Renz W (2008) Engineering environment-mediated multiagent systems. Springer, New York

Toth P, Vigo D (2002) Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discret Appl Math 123(1–3):487–512

Tseng YC, Ni SY, Chen YS, Sheu JP (2002) The broadcast storm problem in a mobile ad hoc network. Wirel Netw 8(2/3):153–167

Vinyals M, Rodrguez-Aguilar JA, Cerquides J (2011) A survey on sensor networks from a multiagent perspective. Comput J 54(3):455–447

Viroli M, Casadei M, Montagna S, Zambonelli F (2011) Spatial coordination of pervasive services through chemical-inspired tuple spaces. ACM Trans Auton Adapt Syst 6:14:1–14:24

Weyns D, Boucké N, Holvoet T (2006) Gradient field-based task assignment in an agv transportation system. In: AAMAS '06: Proceedings of the fifth international joint conference on autonomous agents and multiagent systems. ACM, New York, pp 842–849

Wolpert L, Jessell T, Lawrence P, Meyerowitz E, Robertson E, Smith J (2007) Principles of sevelopment. 3rd edn. Oxford University Press, Oxford

Ye J, McKeever S, Coyle L, Neely S, Dobson S (2008) Resolving uncertainty in context integration and abstraction. In: ICPS' 08: Proceedings of the international conference on pervasive services. ACM, pp 131–140

Yi Y, Gerla M (2003) Efficient flooding in ad hoc networks: a comparative performance study. In: Proceedings of the IEEE international conference on communications, ICC, pp 1059–1063

Journal : Large 11047 Dispatch : 21-4-2012 Pages : 25
Article No. : 9324 ☐ LE ☐ TYPESET
MS Code : ☑ CP ☑ DISK