Description and Composition of Bio-Inspired Design Patterns: the Gossip Case

Jose Luis Fernandez-Marquez, Josep Lluis Arcos, Giovanna Di Marzo Serugendo, Matteo Casadei

***Abstract*—Today's software applications increasingly feature a great deal of openness, dynamism and unpredictable behavior, forcing to shift design and engineering from traditional, centralized approaches to nature-inspired, self-organizing techniques. Among the others, biology has been adopted as a source of inspiration to solve some of the issues proper of nowadays systems by self-organizing techniques, usually exploited in an ad-hoc way. As a result, little or no effort has been made to clearly describe and classify these techniques in terms of design patterns, preventing them from being systematically applied to solve recurrent problems.**

Correspondingly, this paper is targeted at modeling bio-inspired mechanisms in terms of design patterns, arguing that some fundamental biological behavior can play the role of basic design patterns to define higher-level patterns featuring more complex behavior and interaction. In this way, we aim at easing both the creation of new mechanisms from adaptation of existing ones, and the classification of the biological behaviors underlying each pattern. The viability of this approach is exemplified through the description of two bio-inspired mechanisms, aggregation and spreading, taken as basic design patterns to define gossip as a composite design pattern.

***Keywords*-self-organization; bio-inspired design patterns; aggregation; spreading; gossip.**

I. Introduction

Today's software applications increasingly rely on wireless devices – such as, PDAs, laptops, mobile phones or sensors – interacting and aggregating with one another on top of novel infrastructures (e.g. sensor networks, ad-hoc networks). Such infrastructures are characterized by a great deal of openness, dynamism and unpredictability, which cannot be coped with by traditional, centralized approaches to system design and engineering. Accordingly, a paradigm shift is needed in order to cope with these issues and effectively design and engineer system behavior by relying on self-* approaches. Furthermore, as today's infrastructures are characterized by a large number of connected devices with a low computation power, current approaches tend to reduce the computational requirement of algorithms and strengthen interaction and coordination, so as to achieve the intended goals in a collaborative way. As a consequence, new mechanisms are needed that are able to efficiently scale with the aforementioned requirements.

Biological systems have been adopted as a source of inspiration to solve these issues in a self-organizing way. A variety of self-organizing, bio-inspired mechanisms have been applied in different domains, achieving results that go beyond traditional approaches [1]. Researchers usually apply these mechanisms in an ad-hoc way. However, their interpretation, definition, boundary and implementation typically vary among the existing literature, thus preventing these mechanisms from being applied clearly and systematically to solve recurrent problems. A few proposals aimed at expressing some of these bio-inspired mechanisms under the form of design patterns have been provided [2]. These patterns provide useful descriptions that help clarify the definitions of these mechanisms. However, these efforts are still fragmented: no clear catalogue of these patterns is provided, interpretations vary among authors, the relations among patterns and their precise boundaries are not described.

In this paper, we focus on modeling design patterns inspired from biology, arguing that some mechanisms can be described in terms of basic ones, i.e. fundamental biological mechanisms that can be used alone or as a part of more complex patterns. From the one hand, structuring design patterns in such a fashion allows a better support to create new mechanisms and adapt existing ones to solve new problems. On the other hand, this structure also allows for a clear identification and separation of the mechanisms proper of each pattern. Every pattern is provided with a detailed description of the problem, the corresponding solution specified as a set of abstract transition rules, the list of the entities involved in the pattern and their behavior (interaction dynamics and algorithmic behavior).

This work has to be regarded as part of a larger work aimed at providing a comprehensive set of patterns. Due to lack of space, this paper only describes in detail three bio-inspired patterns: aggregation, spreading, and gossip as a composition of the first two patterns. Their exact boundaries and relations are also clearly introduced.

The rest of the paper is organized as follows. Section II introduces related work. Section III presents the reference model afterwards adopted to describe the example patterns. Then, Section IV incrementally details the example patterns, starting from aggregation and spreading, concluding with gossip as a composition of the first two patterns. Finally, Section V concludes, providing for final remarks.

Jose Luis Fernandez-Marquez and Josep Lluis Arcos are with the IIIA, Artificial Intelligence Research Institute, CSIC, Spanish Scientific Research Council, Campus Universitat Autonoma de Barcelona, Catalonia, Spain (e-mail: {fernandez,arcos}@iiia.csic.es.

Giovanna Di Marzo Serugendo is with the Centre Universitaire d'Informatique, Université de Genève, Switzerland (e-mail: Giovanna.DiMarzo@unige.ch).

Matteo Casadei is with the Dipartimento di Elettronica Informatica e Sistemistica, Alma Mater Studiorum – Università di Bologna, Italy (e-mail: m.casadei@unibo.it).

II. Related work

The idea of engineering self-organizing systems has attracted different researchers since 2004. Nagpal et al. [3] present a set of biologically-inspired primitives that describe how organizing principles from multi-cellular organisms may apply to multi-agent systems. That paper was a first attempt towards assembling a catalog 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 where the primitives can be applied. It is then difficult to use them in a systematic way. Mamei et al. [1] propose a taxonomy to classify self-organizing mechanisms and describe a set of mechanisms. Even when these descriptions can drive the implementation of the mechanisms, they are far away to be considered as patterns to apply systematically.

Since 2007, different authors have focused on proposing descriptions of self-organizing mechanisms under the form of software design patterns [4]. 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. [5] propose a set of design patterns for self-organizing systems all related with the ant colonies behavior, together with the idea that a mechanism can be composed from other mechanisms that can be used alone. The provided model, however, presents too many constraints to be generalized and the examples of usage are not related to engineered self-organizing systems. Based on the set of mechanisms proposed in [1], Sudeikat et al. [6] discuss how intended multi-agent systems (MAS) dynamics can be modeled and refined to decentralized MAS designs, proposing a systematic design procedure that is exemplified in a case study. De Wolf [7] presents an extended catalogue of mechanisms as design patterns for self-organizing emergent applications. The patterns are presented in detail and can be used to systematically apply them for engineering self-organizing 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.

III. 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 specialized for the artificial world where the patterns will be engineered.

In biological systems, used as inspiration for self-organizing mechanisms, 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 agents and can produce changes in the system (e.g. toxic clouds, stormy, thunder, a fire). 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 organism are living, change the food location, remove food, add new food, etc.). The communication between the organisms can be direct (e.g. dolphins sending ultrasounds through the water, or beavers emitting sounds to alert about a predator presence, etc.) or indirect using the environment to deposit information that other agents can sense (e.g. pheromone in ants colonies, morphogens in the specialization of cells, etc).

The biological model may be summarized by two layers: organisms and environment, see Figure 1 (a). In order to create a computational model inspired by the biological model, a new layer is added, Figure 1(b). This new layer called the *infrastructure* layer, is necessary because, in an engineered system, the software agent must be hosted in a device with 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 pro-active software entities, (b) the *infrastructure*, that contains *hosts* with computational power, sensors and actuators and (c) the *environment*, the 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' behavior and a place where information can be stored and possibly read by other agents. In most of the 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 a software embedded in it, called *Infrastructural Agent (IA)*. Both IA's and agent's behaviors must be designed to follow self-organizing 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

Mobile Agents	Mobile Hosts	Known Application
no	no	Sensor Networks
no	yes - controlled	Swarm Robotics
yes	no	Sensor Networks
yes	yes - uncontrolled	Pervasive Scenarios

Table I
APPLICATION EXAMPLES - DIFFERENT KINDS OF AGENTS AND HOSTS AND CORRESPONDING KNOWN SYSTEMS.

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. Table I summarizes the different kinds of systems that can be modeled. 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 [8], 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 [9].

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

- **Agents**: autonomous and pro-active software entities running in a host.
- **Infrastructure**: the infrastructure is composed by 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 behaviors present in nature, such as diffusion, evaporation, aggregation, etc.
- **Environment**: The Environment is the 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.

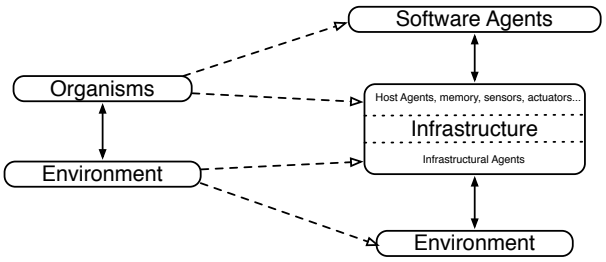


Figure 1. Relevant entities of the biological and computational models.

In this paper, we regard a system as composed of Agents, Infrastructure, Infrastructural Agents, Hosts, and Environment. Behavior 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.

Transition Rules: the behavior of agents or infrastructural agents is defined by a set of rules. We define abstract transitions rules that apply concurrently within a given mechanism. A transition rule is specified by Equation 1. A transition rule has an effect on information, transforming some specified input into some output. A matching input causes the transition to fire instantaneously at a specified frequency noted on top of the arrow (*freq*).

$$name :: input\_value \xrightarrow{freq} output\_value \qquad (1)$$

IV. EXAMPLES OF BIO-INSPIRED DESIGN PATTERNS

In software engineering, a design pattern describes a reusable solution for a commonly recurring problem. Software design patterns were proposed by [4], [10] and [11] for the development of object-oriented software.

Many bio-inspired self-organizing mechanisms have been proposed in literature that allow to achieve a good performance when coping with openness, unpredictability, and dynamism proper of today's decentralized and distributed application domains. However, the knowledge and experience on how, when, and where to use them is spread across the corresponding literature. This motivated to focus on novel researches targeted at proposing a scheme similar to those presented in [7] and [5] to describe design patterns

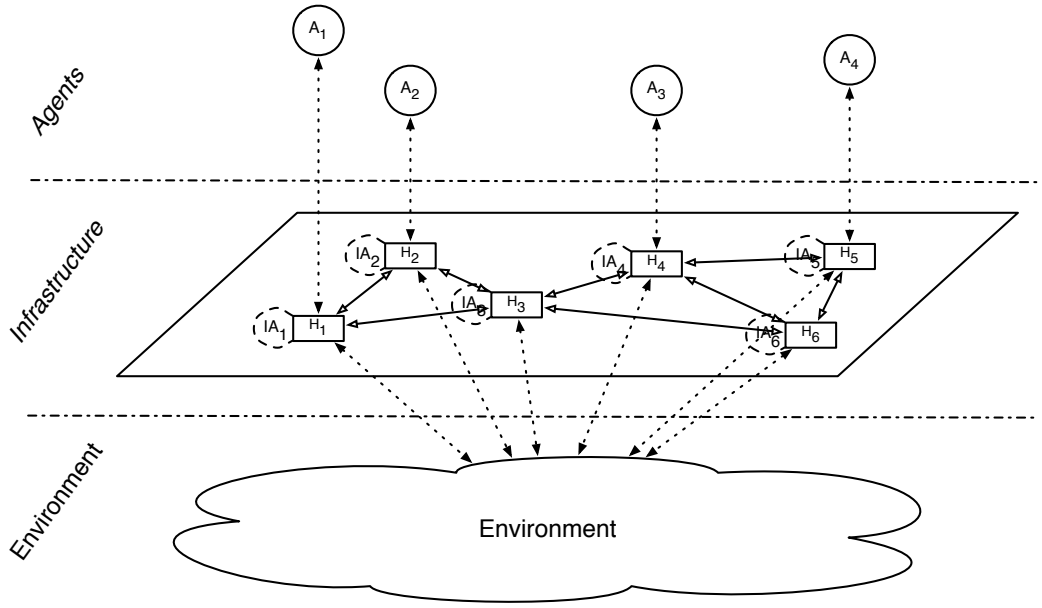


Figure 2. Model

in self-organizing systems. To describe patterns, we use the scheme shown in Table II, which is based on the scheme proposed in [5], extended with some ideas from [7].

The rest of the section is focused on describing aggregation and spreading as examples of fundamental design patterns exploiting bio-inspired behavior observed in nature. These patterns, detailed according to the model introduced in Section III, are then exploited as the basic blocks to define the gossip pattern shown in IV-C.

A. Aggregation Pattern

The Aggregation Pattern is a basic pattern for information fusion. The dissemination of information in large scale systems deposited by the agents or taken from the environment may produce network and memory overload, thus, the necessity to synthesize the information. The Aggregation Pattern reduces the amount of information in the system and assesses meaningful information. It was proposed in [5].

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 assess meaningful information.

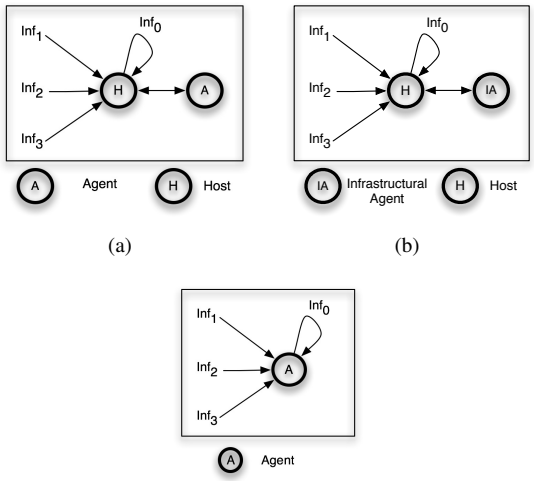
Solution: aggregation consists in locally applying an aggregation operator to process the information and synthesize macro information. This operator can take many forms, such as filtering, merging, aggregating or transforming.

Inspiration: in the 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 done by the environment. Even when there are no agents present in the system, the environment keeps doing the aggregation process.

Forces: aggregation applies on all the information available locally or only on part of that information. The parame-

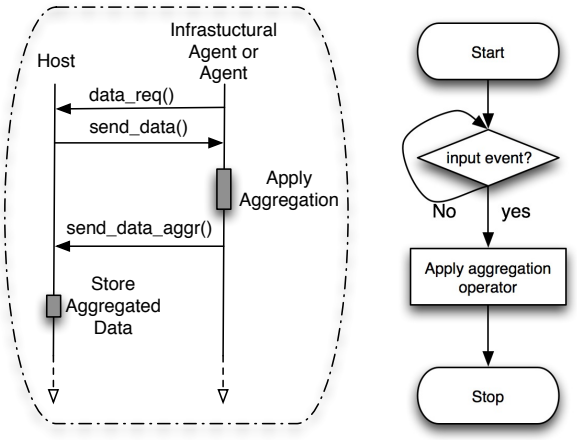
Name	The pattern's name.
Aliases	Alternative names used for the same pattern.
Problem	Which problem is solved by this pattern and situations where the pattern may be applied.
Solution	The way the pattern can solve the problems.
Inspiration	Biological process that inspires the design pattern.
Forces	Prerequisites 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 do the entities of the pattern collaborate to achieve the goal. Typical scenario describing the run-time behavior of the pattern.
Infrastructure	Infrastructural requirements to apply the pattern.
Example	A simple and abstract example of the pattern usage.
Implementation/ Simulation	Hints of how the pattern could be implemented. Include parameters that must be tuned.
Known Uses	Examples of application 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 combine with this pattern or present conflicts with this pattern.

Table II
DESCRIPTION FIELDS



(a) (b) (c)

Figure 3. Aggregation dynamics.



(a) Interaction (b) Flow

Figure 4. Aggregation: agent behavior.

ter involved is the amount of information that is aggregated; it relates to the memory usage in the system. This pattern is not repetitive (i.e. there is no frequency involved, the pattern applies only once), even though it can be repeatedly invoked from within another pattern.

Entities-Dynamics-Environment: aggregation is executed either by agents or by infrastructural agents. In both cases, the agents aggregate the information that they access locally. The information comes from the environment or from other agents. Information that comes from the environment is typically read by sensors (e.g. temperature, humidity, etc.). According to the model presented in Section III, the aggregation is executed by an agent that receives information from the host where the agent is residing. Such a host is either a sensor reading information from the environment or a communication device receiving information from neighboring hosts. Figure 3(a) shows the case of an agent aggregating information, while Figure 3(b) shows the case of aggregation performed by an infrastructural agent. More generally, the aggregation may be applied by any agent that receives information independently of the underlying infrastructure. This general case is shown in Figure 3(c), where the host is abstracted. The aggregation process is not repetitive and finishes when one agent executes the aggregation function.

The transition rule for aggregation is given by Equation 2. Information in input (possibly a set of information) I is transformed into a new set of information through an aggregation operator $op()$.

$$aggregation :: I \rightarrow op(I) \tag{2}$$

Implementation: available information takes the form of a stream of events. Aggregation of information can take various forms: from a simple operator (sum, mult,...) like in ACO, to more complex operators (e.g. Kohonen Self-Organizing Map to aggregate sensor data in clusters [12]). Aggregation operators are classified into four different groups [13]: *(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. input are GPS coordinates and output is the country where the position is 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. Figure 4(a) shows how the agent or infrastructural agent uses the interface provided by the host to get the data, applies an aggregation operator and deposits the aggregated data back in the host. This interaction between the agent or infrastructural agent and the host is shown in Figures 3(a) and Figure 3(b). The flow chart 4(b) shows that the aggregation process starts when the agent receives the information (an event), it then applies the aggregation operator and sends the aggregated information back to the host.

Known uses: aggregation has been used in the ACO algorithm [14], where aggregation is used to aggregate pheromones, simulating higher concentrations when two or more pheromones are close to each other. In [15] the aggregation is also used in digital pheromones for autonomous coordination of swarming UAVs. Moreover, aggregation has been used in the field of information fusion, which studies how to aggregate individual belief bases into a collective one [16], or for truth-tracking in MAS [17].

Consequences: aggregation increases the efficiency in networks (e.g. sensor networks, ad-hoc or P2P), by reducing the number of messages, thus, increasing the battery life. 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.

Related Patterns: the Aggregation Pattern can be implemented together with Evaporation and Gradient Patterns so as to form digital pheromone [15]. The evaporation can be used with aggregation in order to aggregate those information recently collected from the environment. The Gossip Pattern (Section IV-C) is a pattern composed by the Aggregation Pattern and the Spreading Pattern (Section IV-B).

B. Spreading Pattern

The Spreading Pattern is based on direct communication among agents for progressively sending information over the system. The spreading of information in multi-agent systems allows the agents to increment the global knowledge of the system.

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

Aliases: spreading is also known as diffusion, dissemination, flooding, broadcast, epidemic spreading.

Solution: a copy of the information (received or held by an agent) is sent to neighbors 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 proposed as a low level pattern that is extended by all the other patterns that use direct communication. Spreading appears in important processes, such as, Morphogenesis, Chemotaxis and patterns of animal coats (e.g. stripes).

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 changing continuously 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 in that way the number of messages. Another way to reduce the number of messages is to determine the number of neighboring nodes that receive the information. It was demonstrated that it is not necessary to send the information to all the neighboring nodes [18] in order to ensure that every node has received the information.

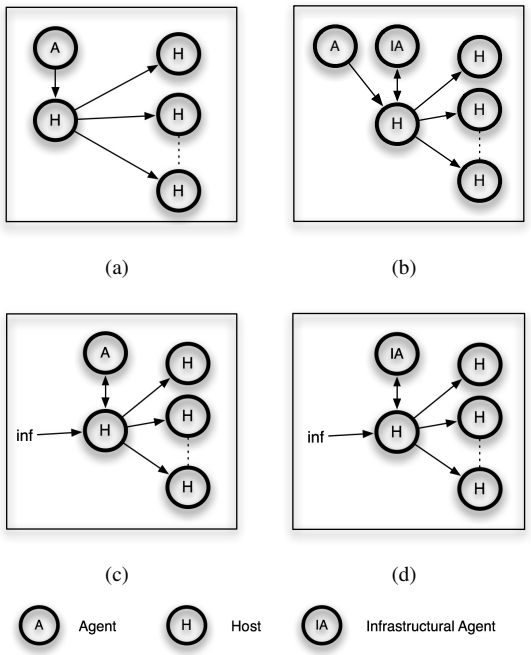


Figure 5. Spreading dynamics.

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. Two cases occur, either the agent spreads itself the information further by sending it to its neighbors, Figure 5(a), or an infrastructural agent is responsible for the spreading, Figure 5(b). If the neighboring node hosts an agent, that agent propagates the information further, Figure 5(c). Otherwise, if the host has no agent, the infrastructural agent is responsible to further propagates the information to the neighbors, Figure 5(d). Each agent forwards the information received to a specified number of neighbors and up to the specified number of hops. The dynamic is usually extended in order 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).

The transition rule for the Spreading Pattern is given by Equation 3. Information in input *inf* is sent to a set of neighbors.

$$spreading :: inf \rightarrow send(inf, neighbors) \qquad (3)$$

Example: Figure 6 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.

Implementation: the most common algorithm used to

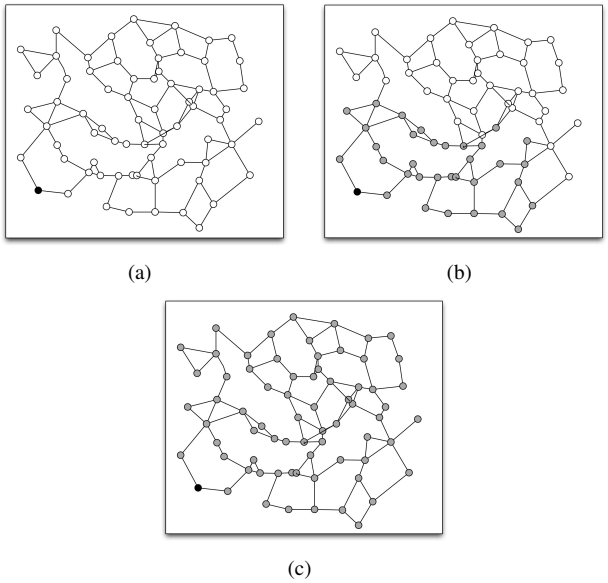
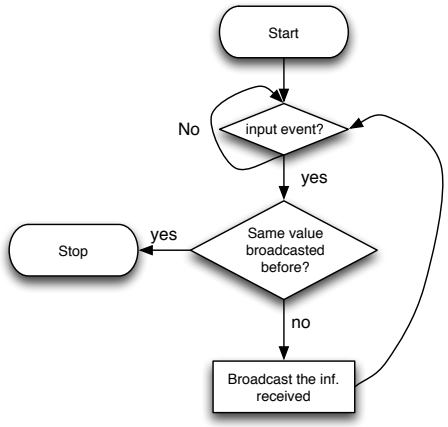


Figure 6. Sample spreading evolution.

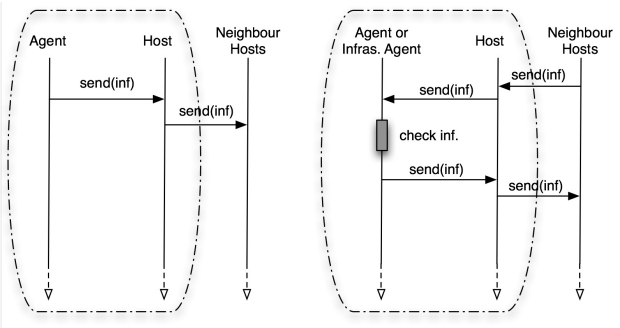
spread the information to the neighbors is the broadcast algorithm. Its implementation may assume a MAC identification phase, in which the protocol exchanges the mac address of two nodes before sending any information, or it may happen without establishing the communication. When the MAC identification is avoided and the broadcast is used, this causes what is called the Broadcast Storm Problem [19]. The Broadcast Storm Problem appears when the radius of 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 optimized broadcast can be implemented. If we implement the broadcast assuming a MAC identification phase, we avoid the Broadcast Storm Problem. This, however, involves a higher number of messages and as a consequence, higher power consumption. The implementation assuming a MAC discovery phase ensures good communication but involves an increment in the number of messages. If the MAC discovery phase is avoided, the Broadcast Storm Problem appears [19] thus forcing the use of optimized broadcast algorithms.

Figure 7(a) shows the flow chart where the information is spread when it is received, provided it has not been received already. Figure 7(b) shows the interaction diagram of the spreading initialization, related with the dynamics shown in Figures 5(a) and 5(b). Figure 7(c) represents the interactions when the information arrives from a neighbor. It is related with the dynamics shown in Figures 5(c) and 5(d).

Known uses: spreading is used in higher level patterns, such as, Gradient, Morphogenesis, or Chemotaxis Patterns. The Spreading mechanism has been applied to several applications: from Swarm motion coordination, to coordination



(a) Agent behavior.



(b) Initialization. (c) Interactions.

Figure 7. Spreading: agent behavior (a) and corresponding initialization (b) and interactions with its host and neighboring hosts (c).

in games, to problem optimization, etc.

Consequences: when the Spreading Pattern is applied, the agents in the system sense information from beyond their local senses. There is then an increment in the network load. This increment becomes extreme when the environment is very dynamic and the agents desire to keep the information updated as soon as possible.

C. Gossip Pattern

The goal of the Gossip Pattern is to obtain an agreement about the value of some parameters in the system in a decentralized 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 neighbors' knowledge and by spreading this aggregated knowledge. Gossip was proposed as an Amorphous computing primitive mechanism by Abelson et al. [20]. Gossip is also known as epidemic communication.

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

Solution: the gossip mechanism combines the aggregation

mechanism (see section IV-A) with the spreading mechanism (see section IV-B) to progressively reach an agreement taking into account the information of all the agents in the system. Information spreads to neighbors, 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 behavior 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 by the Spreading and Aggregation Patterns. It thus presents the same tradeoffs (see Section IV-B and Section IV-A). 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, optimized broadcast can be applied (e.g. not all the neighbors receive the information). The number of neighbors that receive the information is the tradeoff of this pattern. The more neighbors that receive the information, the more robust the system is in the case of failures, but the more network overload is produced.

Entities-Dynamics-Environment: the entities involved in the gossip mechanism are agents, infrastructural agents and hosts. As it was presented before, gossip is a combined 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 set of its neighbors. If an agent is hosted in one of those neighbor nodes, the agent gets the information, aggregates the information received with its own information and resends the aggregated information to its own neighbors nodes, Figure 8(a). The same behavior is produced by the infrastructural agent when no agent is hosted in one host and the host receives an information, Figure 8(b). The process can be generalized (abstracting the host) as shown in Figure 8(c), where an agent receives the information from other agents, aggregates the information received with its own information and then, sends the aggregated information to the neighbors. 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.

The transition rule for gossip is given by Equation 4. Information received from the neighbors (I_{rcvd}) is aggregated to local information (inf_{local}) and sent to a set of neighbors.

$$gossip :: (I_{rcvd}, inf_{local}) \rightarrow send(op((I_{rcvd}, inf_{local})), neighbors) \quad (4)$$

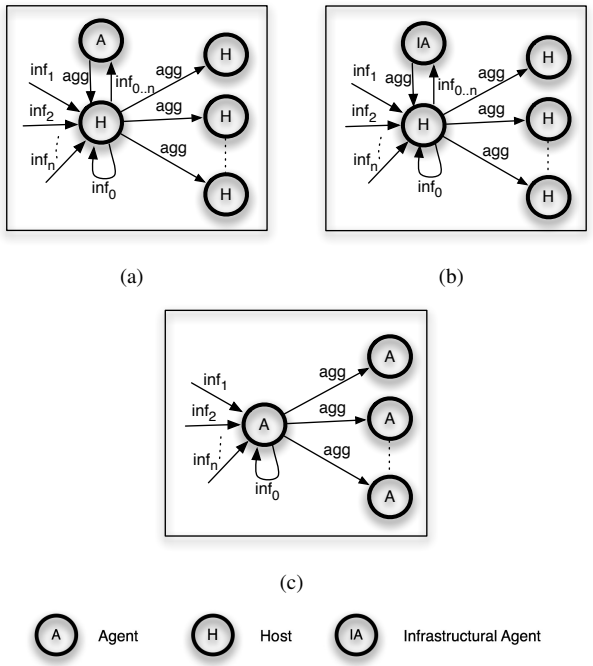
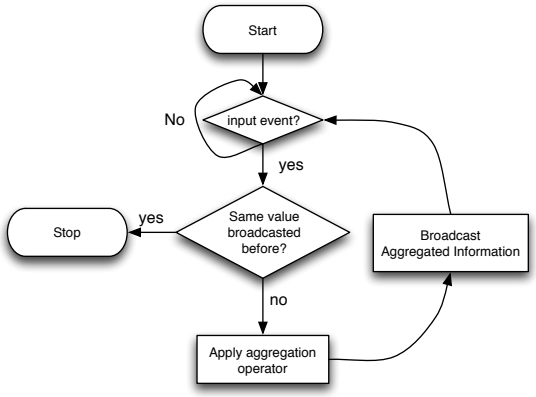


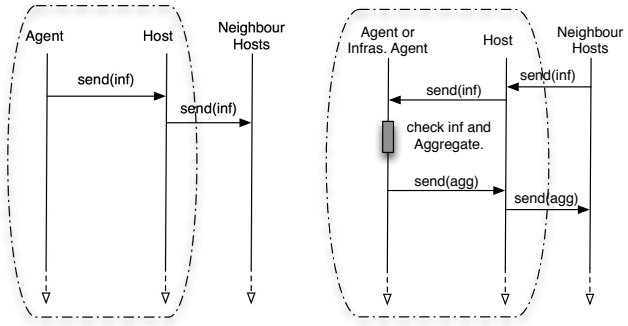
Figure 8. Gossip dynamics.

Implementation: regarding implementation, optimized broadcast can be applied. One interesting example of implementation appears in [21], 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 optimization decrements the number of messages by 35%. Figure 9(a) shows the flow chart for the standard gossip mechanism where the information is spread using the broadcast. Figures 9(b) shows the interaction between the agent that initiates the gossip process, the host where the agent is running and the neighbor hosts. Once the gossip has started, the agents and infrastructural agents follow the behavior presented in Figure 9(c).

Known uses: Kempe et al. [22] analyze a simple gossip-based protocols for the computation of sums, averages, random samples, quantiles, and other aggregate functions, and demonstrate that this protocol converges to the agreement faster than the uniform gossip. Norman et al. [23] propose an aggregation based on Evolutionary Algorithm. They present a mechanism for coordination in 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 scenario 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



(a) Agent behavior.



(b) Initialization. (c) Interactions.

Figure 9. Gossip: agent behavior (a), initialization (b) and interactions with the host and neighboring hosts (c).

conventions.

Consequences: the main advantage of gossip is the robustness: even in the presence of failures, the pattern is able to reach the agreement.

Related Patterns: the Gossip Pattern is composed by the Spreading Pattern (see Section IV-B) and the Aggregation Pattern (see section IV-A).

V. Conclusion and Future Work

This paper is a first attempt towards a clear classification of bio-inspired design patterns targeted at easing the design and engineering of today's application domains. The contribution of the paper is twofold. From the one hand, we proposed a concrete model upon which to define patterns, that relies on the concept of structured patterns, i.e. the fact that some basic patterns can be exploited as building blocks for defining more complex patterns. On the other hand, we provided an example of how the model can be concretely adopted by presenting aggregation and spreading as basic bio-inspired patterns, exploited as a base to define gossip as a composite patterns.

As a future work, we are planning to work on a comprehensive description of the most relevant bio-inspired patterns found in existing literature, as well organize them into layers according to their biological inspiration and role in defining composite patterns. Furthermore, as a part of the EU Project SAPERE, we will investigate the role of these bio-inspired patterns to enact emergent and self-organizing behaviors into a middleware target at developing service-based pervasive applications in novel domains.

Acknowledgments

Work supported by the EU SAPERE Project under contract no.256873 and EVE under contract TIN2009-14702-C02-01. The first author holds a FPI scholarship from the Spanish Government.

References

[1] M. Mamei, R. Menezes, and F. Tolksdorf, R.and Zambonelli, "Case studies for self- organization in computer science." *J. Syst. Archit*, vol. 52, pp. 433–460, 2006.

[2] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes, "Design patterns from biology for distributed computing," *ACM Trans. on Autonomous and Adaptive Sys*, vol. 1, pp. 26–66, 2006.

[3] R. Nagpal, "A catalog of biologically-inspired primitives for engineering self-organization. engineering self-organising systems: Nature-inspired approaches to software engineering (2003); 2977: 5362," in *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering. Volume 2977 of Lecture Notes in Computer Science*. Springer, 2004, pp. 53–62.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley, 1995.

[5] L. Gardelli, M. Viroli, and A. Omicini, "Design patterns for self-organizing multiagent systems," in *In Proceedings of EEDAS*, 2007.

[6] J. Sudeikat and W. Renz, "Engineering environment-mediated multi-agent systems," D. Weyns, S. A. Brueckner, and Y. Demazeau, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Toward Systemic MAS Development: Enforcing Decentralized Self-organization by Composition and Refinement of Archetype Dynamics, pp. 39–57.

[7] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," *Engineering Self-Organising Systems*, vol. 4335, pp. 28–49, Feb. 2007.

[8] J. Fernandez-Marquez, G. Di Marzo Serugendo, and A. J. L., "Infrastructureless spatial storage algorithms," *ACM Transactions on Autonomous and Adaptive Systems (TAAS), (In press)*, 2011.

[9] M. Vinyals, J. A. Rodríguez-Aguilar, and J. Cerquides, "A survey on sensor networks from a multiagent perspective," *The Computer Journal*, 2010, in press. Published on-line on February 2010. DOI:10.1093/comjnl/bxq018.

[10] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture: A system of patterns*. John Wiley & Sons, 1996.

[11] J. Lind, "Patterns in agent-oriented software engineering," *Agent-Oriented Software Engineering III*, vol. 2585, pp. 47–58, Jul. 2002.

[12] S. Lee and T.-C. Chung, "Data aggregation for wireless sensor networks using self-organizing map," in *AIS*, 2004, pp. 508–517.

[13] G. Chen and D. Kotz, "Context aggregation and dissemination in ubiquitous computing systems," in *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, ser. WMCSA '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 105–. [Online]. Available: http://portal.acm.org/citation.cfm?id=832315.837559

[14] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," *New Ideas in Optimization*, pp. 11–32, 1999.

[15] H. Parunak, M. Purcell, and R. Connell, "Digital pheromones for autonomous coordination of swarming uavs," in *In The First AIAA Unmanned Aerospace Vehivales, Systems, Technologies, and Operations*. American Institute of Aeronautics and Astronautics, 2002.

[16] E. Grégoire and S. Konieczny, "Logic-based approaches to information fusion," *Inf. Fusion*, vol. 7, no. 1, pp. 4–18, 2006.

[17] G. Pigozzi and S. Hartmann, "Aggregation in multi-agent systems and the problem of truth-tracking," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 07), 1418 May 2007, Honolulu, Hawaii, USA*, 2007, pp. 674–676.

[18] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst.*, vol. 17, pp. 41–88, May 1999. [Online]. Available: http://doi.acm.org/10.1145/312203.312207

[19] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wirel. Netw.*, vol. 8, no. 2/3, pp. 153–167, 2002.

[20] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, J. Thomas F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss, "Amorphous computing," *Commun. ACM*, vol. 43, no. 5, pp. 74–82, 2000.

[21] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-based ad hoc routing," *IEEE/ACM Trans. Netw.*, vol. 14, no. 3, pp. 479–491, 2006.

[22] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pp. 482 – 491, oct. 2003.

[23] N. Salazar, J. A. Rodriguez-Aguilar, and J. L. Arcos, "Robust coordination in large convention spaces," *AI Communications*, vol. 23, no. 4, pp. 357–372, 2010. [Online]. Available: http://dx.doi.org/10.3233/AIC-2010-0479