2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems

Augmenting the Repertoire of Design Patterns for Self-Organized Software by Reverse Engineering a Bio-Inspired P2P System

Paul L. Snyder, Giuseppe Valetto
Drexel University
Department of Computer Science
Philadelphia, Pennsylvania, USA

Jose Luis Fernandez-Marquez,
Giovanna Di Marzo Serugendo
University of Geneva, CUI
Carouge, SWITZERLAND

***Abstract*—Investigations of self-organizing mechanisms, often inspired by phenomena in natural or societal systems, have yielded a wealth of techniques for the self-adaptation of complex, large- and ultra-large-scale software systems.**

The principled design of self-adaptive software using principles of self-organization remains challenging. Several studies have approached this problem by proposing design patterns for self-organization. In this paper, we present the results of applying a catalog of biologically inspired design patterns to Mycoload, a self-organizing system for clustering and load balancing in decentralized service networks.

We reverse-engineered Mycoload, obtaining a design that isolates instances of several patterns. This exercise allowed us to identify additional reusable self-organization mechanisms, which we have also abstracted out as design patterns: SPECIALIZATION, which we present here for the first time, and a generalized form of COLLECTIVE SORT. The pattern-based design also led to a better understanding of the relationships among the multiple self-organizing mechanisms that together determine the emegent dynamics of Mycoload.

***Keywords*-self-organization; design patterns; bio-inspired algorithms; design modeling.**

I. INTRODUCTION

Modern computing and communications systems continue to expand in scale. We witness more and more examples of ubiquitous computing systems, social networks, wireless sensor networks, peer-to-peer overlays, and many others, which encompass huge numbers of components on heterogeneous devices, often under multiple ownerships.

While techniques of self-organization have proven effective in enabling system–wide adaptations of large-scale software in such domains, it remains challenging to represent and to reason about these mechanisms [1].

To address this problem, several attempts have been made to identify and collect design patterns for self-organization [2], [3]. Previous work by Fernandez-Marquez *et al.* [4] has proposed a catalog of bio-inspired design patterns, defining a hierarchy of patterns where basic patterns are used as elements of composed, higher-level dynamics. The motivation of that work is to organize the growing body of knowledge in the area, and foster modular, reusable design of self-organizing software.

To explore the practical implications and benefit of those patterns, we modeled the design of a decentralized service network, called Mycoload [5], running on top of a biologically inspired peer-to-peer network developed by Snyder *et al.* [6]. This reverse engineering exercise has yielded multiple contributions: (1) it has made evident how a number of mechanisms that we have implemented in Mycoload are instances of the patterns in our catalog, demonstrating how design pattern abstractions can provide a greater understanding of a real-world, complex software system that uses self-organization principles for its self-adaptation; (2) it has shown how pattern-based decomposition can expose otherwise implicit interactions among those patterns, interactions that play an important role in determining the overall dynamics of a system; and (3) it has allowed the isolation of other reusable self-organizing mechanisms that we can now introduce in our catalog as design patterns. Due to space limitations, this paper focuses on the new patterns identified. The full decomposition and discussion can be found in [7].

II. RELATED WORK

Researchers have repeatedly taken inspiration from natural self-organizing systems, and have identified adaptive mechanisms that can be mimicked in computing systems. This approach allows results that often go beyond the possibilities of centralized control in many scenarios [8], [9]. However, these self-organizing mechanisms are typically approached in an ad-hoc or highly application-specific manner, which prevents their systematic reuse, and hampers their application to recurrent problems across different domains.

Among the works that attempt to define design patterns for self-organizing software, some focus on the discovery and definition of a single pattern [10], [11]; others propose concrete implementation descriptions [12]; yet others catalog multiple patterns [3], [13]. Our previous work [4], discussed in Section III, is most similar to the latter. One feature that sets our catalog of bio-inspired patterns apart from other efforts is the organization of the patterns into layers, and the documentation of composition relationships between patterns in those layers. As pointed out by Parunak and Brueckner [1], the definition of composition and decompo-

978-0-7695-4851-7/12 $26.00 © 2012 IEEE
DOI 10.1109/SASO.2012.23

sition relationships is important, because—while it seems clear that certain self-organization mechanisms can be obtained from finer-grained ones—which ones should be used as primitives, and how they should be combined is often not clear, and almost never explicitly codified. Some interesting work on composition has been performed by Sudeikat and Renz [14], who take the approach of looking for congruent feedback loops among multiple self-organizing mechanisms.

In general, there is still a lack of guidance on how to use identified self-organizing design patterns in the engineering of larger self-adaptive systems that rely on multiple mechanisms. The work we present with respect to the reverse engineering of Mycoload may be the first that tries to apply patterns exhaustively to model an existing complex self-organized system, in order to shed a better light on its self-adpative dynamics. In that sense, it is similar to the work by Ramirez *et al.* [15], which analyzed a list of adaptation design patterns, and re-engineered an application using those patterns. The authors could identify several advantages of using such design patterns. Since that work focused on adaptation design patterns, we consider that a similar effort must be done also for specifically evaluating self-organizing design patterns, and fostering their reuse.

III. Self-Organization Design Patterns

In previous work, Fernandez-Marquez *et al.* [4], presented a catalog of self-organization design patterns (largely inspired from mechanisms observed in biology), and analyzed the relations between them. One contribution is a three-layer classification (basic, composed and high-level patterns), shown in Figure 1. This classification scheme also includes relations among different self-organizing mechanisms, such as composition and usage: patterns in the lower layers provide building blocks for more sophisticated patterns at the higher layers. A good example is the Digital Pheromone pattern, which uses a Spreading mechanism to disperse the pheromones over the environment, an Aggregation mechanism to combine multiple pheromone concentrations at a location, and an Evaporation mechanism to cause pheromone concentrations to decay over time.

The catalog shows how a limited number of basic mechanisms are at the basis of a large set of powerful self-organizing dynamics that have been examined in the literature (including gossip, gradients, and morphogenesis), which can also be expressed as design patterns. Our classification scheme thus offers a key to understand complex as well as basic self-organization mechanisms. Most importantly, it enables the design of self-organizing applications in terms of well-defined, modular and reusable blocks.

IV. Pattern-Based Reverse Engineering of Mycoload

Mycoload [5] is a self-organizing system for clustering and load-balancing in unstructured peer-to-peer networks.

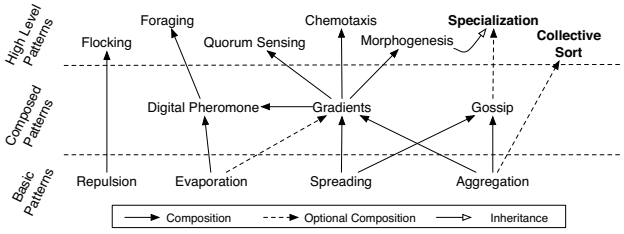

Figure 1: Self-Organization Design Patterns

Mycoload is an extension to Myconet [6], a protocol for constructing superpeer-based overlays, inspired by the growth patterns of fungal root systems. It rapidly self-heals, repairing damage caused by failed peers, and dynamically adjusts the network topology to changing conditions. One of its distinguishing characteristics is that superpeers move through a hierarchical set of protocol states. Each state has different roles and responsibilities in maintaining the overlay, and peers are promoted or demoted to those states based on how well they are able carry out those responsibilities.

We reverse-engineered Mycoload using the pattern catalog from [4] in order to decompose and identify the multiple self-organizing mechanisms that collectively determine the system's dynamics. The system has four main functional areas: discovery of non-neighbor peers; exploitation of heterogenous peer capabilities and optimization of their roles within the overlay; collection of peers providing the same type of services into clusters in the overlay network; and load balancing of jobs between peers.

Across these functional areas, we identified several instances of self-organization patterns: the low-level Spreading, Aggregation, and Evaporation patterns, as well as the higher-level Gossip and Gradient patterns. We also isolated two important aspects of its self-organizing behavior that are not encompassed by other patterns but can themselves become reusable mechanisms. We have abstracted those mechanisms as instances of two design patterns: the first one, Specialization has not been presented as a self-organization design pattern before in the literature, as far as the authors know; the second one is a generalization of the Collective Sort in pattern [2]. Due to space limitations, readers are referred to [7] for the full description of the role of these patterns within the design and dynamics of the Mycoload system.

In Sections V and VI, we present these patterns, following the self-organization design pattern structure used in [4].

V. Specialization Pattern

Specialization is a mechanism widely used in complex systems for achieving improved efficiency by exploiting the natural heterogeneity of the entities taking part in the system. Through Specialization, each individual entity is assigned a specific role depending on its capabilities and contextual local information. A useful survey of specialization in self-organizing systems can be found in [16].

According to the SPECIALIZATION pattern, system entities change the rules under which they operate, depending on features or properties of the entity itself, or contextual information from its environment and neighbors. For example, a computer with high amounts of memory available could store information on behalf of nodes with low memory; a computer with sensors could provide sensed information to other computers; a network node with a large amount of bandwidth could be elected to act as a router for traffic transmitted by other nodes, and so on. The assumption of specialized roles may be influenced by other entities, or may need to be further modulated by information from other self-organizing mechanisms, in order to adapt to changing system conditions and requirements in the system.

Name: SPECIALIZATION

Aliases: None to our knowledge.

Problem: Global optimization of system efficiency by increasing or decreasing the contributions of individual entities or by otherwise changing the rules under which those entities operate.

Solution: Individual entities are assigned a specific role or set of behavioral rules depending on their capabilities and contextual local information. SPECIALIZATION optimizes entities' contributions in order to increase the overall performance of the system.

Inspiration: The specialization process appears in many macro- and micro-level systems. Some examples are the specialization of cells in a human body or the specialization of individual humans to fill particular roles in society.

Forces: Depending on how the contextual information used for making decisions regarding specialization is acquired and which patterns are used in transferring and maintaining this information, different trade-offs can appear. The most common patterns are SPREADING and AGGREGATION (see the forces discussed in their pattern descriptions [4] for more details.) In general, though, the information used by SPECIALIZATION can come from any other self-organizing mechanisms or a combination of those mechanisms, and their dynamics will influence and possibly be mutually influence the resulting specializations.

Entities: The entities participating in the SPECIALIZATION pattern are: (1) software agents that modify their behavior depending on their capabilities (or the capabilities of their hosts) and environmental information (e.g. external requirements); (2) hosts that provide sensors, memory, communication capability, computational power, etc. to the software agents; and finally (3) Environment, all that is external to the hosts (e.g. the space where host are located, external requirements that are injected in the system, etc.. . .)

Dynamics: Agents retrieve contextual information from their own knowledge and from their neighbor agents, or from the environment by using sensors or querying an externally implemented environmental model. To describe the dynamics we use the same notation as in [4], where information contained in the system is modelled as a tuple $\langle L, C\rangle$, where L is the location where the information is stored (possibly within an agent or maintained by an external middleware), 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 resemble chemical reactions between patterns of tuples, 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 (that is, its scheduling policy).

$$\texttt{state\_evolution} :: \langle L, [cInf, State, C]\rangle \xrightarrow{r_{sp}} \langle L, [cInf, State', C]\rangle$$
$$\textit{where } State' = \pi(cInf, State, C)$$

In the above rule, $cInf$ is the contextual information accessible to the agent, $State$ is its previous role before the specialization occurs (*i.e.*, the set of rules under which it operates), $State'$ is the new role (and consequent set of rules) adopted as a result of the specialization process, and π is a function that produces this new state from the given contextual information, current agent state, and any local information.

Environment: The hosts must have different features or the system must display other heterogeneities (for example, in the distribution of agents in different locations) that allow SPECIALIZATION to assign an appropriate role based on those features and the contextual information.

Implementation: We have identified two different implementations: (1) An agent decides to change its role in the system by taking into account the capabilities of the local environment where it resides and acquired contextual information (e.g., the system's requirements); and (2) An agent is positioned to determine that one of its neighbors should adopt a new role. An example of the second case is when one node is providing services to other nodes in the system but it is reaching the maximum number of clients; in such a case the node can replicate the information served to a new node (selected according to some suitability measure from among other nodes in its neighborhood) and target it to assume the role of an additional service provider.

Two types of rules can be used to drive the specialization of agents, determining which role an indivdual adopts depending on its capabilities and context: fixed rules, and adaptive rules. Fixed rules are defined by developers at design time, agents switch among behaviors from a static set. Adaptative rules may be changed by the agents during run time in order to contribute to the optimization of the global system behaviour. Evolutionary approaches have been used in the field of autonomic computing to establish

sets of norms, policies or rules that drive the system to the desired emergent behaviour, even when environmental changes occur. A possible implementation was introduced in [17], which uses a distributed genetic algorithm. In that approach, each agent participating in the system performs local evaluations and adjustments that are then shared with other agents using spreading mechanisms.

Known Uses: Specialization has been used by a large number of self-organizing applications. Examples include: (1) Overlay networks where some nodes decide to become routers based on their available resources and their connectivity with other nearby nodes [18] (2) Aiming to localize diffuse event sources in dynamic environments using large scale wireless sensor networks, agents change their roles in order to locate and track diffuse event sources [19]. (3) To balance the load among nodes with different services, Mycoload [5], builds a superpeer topology where more powerful nodes adopt several different, specialized roles in the creation and maintainance of the overlay. (4) [20] describes a robust process for shape formation on a sheet of identically programmed agents (origami) where the heterogeneity of the agents comes from their location.

Consequences: Specialization locally increases or decreases the contribution of individual nodes, improving the global efficiency of the system.

Related Patterns: In the MORPHOGENESIS pattern, the role of the agents changes depending on their relative positions, typically communicated via a GRADIENT. Thus, MOPHOGENESIS is of the same family but more specific than SPECIALIZATION.

VI. GENERALIZED COLLECTIVE SORT PATTERN

Collective sort is a clustering mechanism that enables segregation or relocation of entities into similar-type groupings within the context of a collection of system elements according to some property of the entities or requirement of the system. Self-organizing algorithms for collective sorting have been developed based on observations of biological phenomena, particularly the processes of brood sorting and cemetery formation by social insects [21].

The mechanism discussed in this pattern is a generalization of the biologically inspired collective sorting proposed as a design pattern for tuple spaces by Gardelli *et al.* [2] and analyzed as an environmental coordination mechanism by Sudeikat and Renz [22]. The usual formulation of the collective sort algorithm assumes a case where active agents relocate inactive data items; the COLLECTIVE SORT pattern presented in this paper extends this to include cases where the agents themselves may be the entities to be grouped, or where different environmental abstractions are being used. These variants are discussed in the Implementation section of the pattern description, below.

Name: COLLECTIVE SORT

Aliases: Brood Sorting, Cemetery Formation, Collective Clustering

Problem: A system contains a number of scattered data or entities that need to be brought into relative proximity with other similar data or entities.

Solution: Individual agents move through an environment, encountering data items as they travel. By picking up and dropping these items based on local heuristics, elements with similar properties are progressively gathered into homogeneous groups or clusters.

Inspiration: Brood sorting and cemetery formation by social insects [21]

Forces: This pattern starts from a disordered arrangement of entities within an environment and progressively reduces that disorder. Thus, it is affected by entity distribution, and other other forces that act to change the location of these entities. The function used by entities to evaluate local density of entities (as well as an entity's range of perception) will affect the outcome of the sort. In particular, a small range may induce the formation of multiple small collections. The choice of rules and probabilities for the picking up and dropping of entities may also influence the speed of convergence or the resulting topological distribution.

Entities: The entities associated with the COLLECTIVE SORT pattern are: (1) data items that have an associated property for which similarity can be assessed; and (2) active agents that are able to examine and relocate data items of type (1). Note that, depending on the implementation, (1) and (2) may be the same entities. For example, the active agents may themselves possess the property that is the subject of sorting, and hence the emergent order will be expressed by the arrangement of the agents themselves.

Dynamics: This pattern relies on three rules which, together, tend to progressively relocate similar elements into similar vicinities: a *Movement Rule* that relocates agents within a set of candidate locations, a *Pick-Up Rule* that connects an agent to an element so it can be moved, and a *Drop Rule* that leaves a held element at a current location. These rules are followed by the active agents. The Movement rule is frequently implemented as random exploration, but could also take advantage of available contextual information if appropriate. The Pick-Up and Drop rules select entities to be clustered when they are encountered, tending to remove elements from areas of high diversity and deposit them in areas of low diversity. Two approaches to applying these general rules are discussed in the Implementation section.

Environment: The environment provides the context within which the proximity of data items is interpreted. Thus, entities must have a concept of location, be able to change location, and be able to detect other nearby entities within that environment; they must also have a means of evaluating the similarity of data items thus detected.

Implementation: Previous descriptions have favored a particular implementation of COLLECTIVE SORT; specif-

ically, they assume the use of a distributed tuple space. The form described here (which emerged from our reverse engineering exercise) encompasses other models, such as the graph-oriented, peer-to-peer environment we discuss.

In the tuple-space formulation, active agents move between spaces, carrying tuples with them. For the movement rule, agents explore the environment and encounter data items as they move. The choice of movement strategy is frequently random, but may also be informed by other information, such as from CHEMOTAXIS.

Agents "transport" data items from place to place, tending to move them from areas with lesser concentration to areas with greater concentration of items of the correct type. This rule may rely on direct observation of data items in a vicinity (as with observation of neighbors in FLOCKING); it may rely on AGGREGATION to estimate the local density of data items of a particular type; or the agent may make its own estimate by maintaining a memory of recently encountered data items. A general way to express how pick-up occurs is through a probabilistic function of the density of data items [23]. See [7] for a more detailed formulation.

In a more general view of COLLECTIVE SORT, an abstract notion of grouping can be used to apply the same strategy in a scenario where the environment is defined by a pattern of neighbor relationships between nodes (composing a graph, as in a P2P network), and where the nodes themselves are labelled with some property (*e.g.*, a node type) upon which clustering should be performed. In this dynamic graph scenario, "movement" is considered to be selecting a candidate location for growing a new neighbor relationship, "picking up" is adding a new neighbor, and "dropping" is severing an existing neighbor relationship.

1. Movement Rule: Agents explore randomly by selecting a potential new neighbor from a set of possible candidates. In many P2P networks, and in Mycoload, this set is maintained by a separate mechanism that implements the GOSSIP pattern, and thus provides each node with fresh samples of candidate non-neighbor nodes. For a node V with neighbor set $N(V)$, a new possible neighbor $Cand$ is selected:

$$movement :: \langle V, N(V)\rangle \xrightarrow{r_{move}} \langle V, N(V), Cand\rangle$$
where $Cand = random(\text{CANDIDATES}(V))$ and $Cand \notin N(V)$

Note that this may also result in V finding a cluster of its own type (if the new neighbor W is of the same type) if it was not already in one, or finding a connecting path for two disconnected same-type sub-clusters.

2. Pick-Up Rule: Once the movement rule has selected a new possible neighbor $Cand$, the agent may add it as a new neighbor to itself

$$pick\_up :: \langle V, N(V), Cand\rangle \xrightarrow{r_{pick\_up}} \langle V, N(V) \cup \{Cand\}\rangle$$

In Mycoload, for example, the pick-up rule will be executed if (a) V does not currently have a neighbor that is its same type, (b) if V has under a certain number of different-type neighbors, or (c) execute anyway with a small probability to prevent the system from settling into a local optimum. Thus, agents will wander until they find a cluster of the same type (whether by encountering it by moving or dropping in place by another agent), but will also try to keep connections to neighbors of other types in order to help other nodes move toward an appropriate cluster. The pick-up rate thus declines as the nodes converge toward clusters.

3. Drop Rule: Dropping for a node V is performed by randomly selecting a neighbor $W \in N(V)$ (where the type of W with neighbor set $N(W)$ is different from the type of V) and, if V also has a neighbor $U \in N(V)$ with the same type as W, by transferring W to become a neighbor of U:

$$drop :: \langle\langle V, N(V)\rangle, \langle W, N(W)\rangle, \langle U, N(U)\rangle\rangle \xrightarrow{r_{drop}}$$
$$\langle\langle V, N(V) - W\rangle, \langle W, N(W) \cup \{U\}\rangle, \langle U, N(U) \cup \{W\}\rangle\rangle$$

Known Uses: Collective sort and brood sorting-inspired approaches to distributed self-organizing systems have been applied to several problems areas: (1) Collecting similar tuples from a distributed tuple system into a single tuple space [24]. Of interest is Casadei *et al.*'s approach using "noise" tuples to implement a simulated annealing-type approach to avoiding local optima [25]. (2) Collective sorting as a coordination mechanism for swarms of self-organized robots, proposed as early as 1991 by Deneubourg *et al.* [26] and extended by later researchers [27]. (3) Storage and retrieval of Semantic Web documents. Presenting such a scenario, Muhleisen *et al.* [28] discuss in particular the role of similarity metric selection in collective sorting. (4) Intrusion detection. Sudeikat and Renz [22] identify brood sorting as suitable for providing a portion of the self-organizing dynamics for a stigmergic IDS. (5) Clustering of same-type nodes in peer-to-peer networks. Mycoload [5] uses a collective sort approach to build clusters of peers offering the same service types; the specific role of the collective sort mechanisms is discussed in this paper.

Consequences: Collective sort enables clustering of data or other entities into groups of similar type. The resulting order may increase efficiency of operations on this data.

Related Patterns: AGGREGATION is often used to estimate local density of data items. SPREADING and GRADIENT may be used to disseminate information about the density of particular kinds of data, and CHEMOTAXIS may be used to guide agent movement. SPECIALIZATION may be used to select specific items among the ones in the resulting groupings for particular roles.

VII. CONCLUSIONS

We have presented two novel self-organization design patterns resulting from the pattern-based reverse-engineering of an entire self-organized software system, Mycoload. This effort also demonstrated the effectiveness of a well-organized catalog of design patterns [4]. An outstanding challenge in this field is understanding the global emergent

behavior that results from multiple interacting mechanisms. A pattern-based design provides leverage for this problem, since the resulting system decomposition highlights the interactions among patterns, and their contribution to the overall self-organizing dynamics.

A benefit of a pattern-based approach is the possibility of identifying additional self-organizing behaviors within the system, which can be themselves be abstracted as reusable mechanisms. Here, we have been able to identify two such modules, the SPECIALIZATION and COLLECTIVE SORT patterns. Both of these capture mechanisms that are present in a number of other self-organized software systems: SPECIALIZATION solves the recurring problem of self-selected differentiation of roles among system elements or agents; and COLLECTIVE SORT enables the organization of disparate data or agents into homogeneous groups.

As the number of self-organization design patterns that are extracted from existing systems increases, and and as we make progress in understanding how they relate and how they can be used together, the process described in this paper holds great promise for providing the engineers of self-organized software with increased insight on the principles around which self-organization mechanisms can be built repeatably and leveraged effectively for the self-adaptation of large- and ultra-large scale systems.

REFERENCES

[1] H. Parunak and S. Brueckner, "Software engineering for self-organizing systems," in *Proc. of 12th Int'l Wkshp. on Agent-Oriented Software Engineering (AOSE 2011)*, 2011.

[2] L. Gardelli, M. Viroli, and A. Omicini, "Design patterns for self-organizing multiagent systems," in *2nd Int'l Wkshp. on Engineering Emergence in Decentralised Autonomic System (EEDAS) 2007*, ICAC 2007, June 2007, pp. 62–71.

[3] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," in *Proc. of 4th Int'l Conf. on Engineering Self-Org. Systems*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 28–49.

[4] J. L. Fernandez-Marquez, G. Di Marzo Serugendo, S. Montagna, M. Viroli, and J. L. Arcos, "Description and composition of bio-inspired design patterns: a complete overview," *Natural Computing*, pp. 1–25, 2012.

[5] G. Valetto, P. L. Snyder, D. J. Dubois, E. D. Nitto, and N. M. Calcavecchia, "A self-organized load-balancing algorithm for overlay-based decentralized service networks." in *Proceeding of SASO'11*. IEEE, 2011, pp. 168–177.

[6] P. Snyder, R. Greenstadt, and G. Valetto, "Myconet: A fungi-inspired model for superpeer-based peer-to-peer overlay topologies," in *SASO'09*, 2009, pp. 40–50.

[7] P. Snyder, G. Valetto, J. L. Fernandez-Marquez, and G. D. M. Serugendo, "Describing self-organizing software with design patterns: A reverse engineering experience," Drexel University, Tech. Rep. DU-CS-12-07, 2012.

[8] R. Nagpal, "A catalog of biologically-inspired primitives for engineering self-organization," in *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering. LNCS Vol. 2977*. Springer, 2004, pp. 53–62.

[9] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli, "Case studies for self-organization in computer science," *Jrnl. of Systems Architecture*, vol. 52, pp. 443–460, Aug. 2006.

[10] H. Kasinger, B. Bauer, and J. Denzinger, "Design pattern for self-organizing emergent systems based on digital infochemicals," in *Proc. of the Int.Conf. on Engineering of Autonomic and Autonomous Systems (EASe'2009)*. IEEE Computer Society, 2009, pp. 45–55.

[11] H. Parunak, S. Brueckner, D. Weyns, T. Holvoet, and P. Valckenaers, "E pluribus unum: Polyagent and delegate MAS architectures," in *Proc. of 8th Int'l Wkshp. on Multi-Agent-Based Simulation (MABS07)*. Springer, 2007, pp. 36–51.

[12] M. H. Cruz Torres, T. Van Beers, and T. Holvoet, "(no) more design patterns for multi-agent systems," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, ser. SPLASH '11 Workshops. New York, NY, USA: ACM, 2011, pp. 213–220.

[13] 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.

[14] 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.

[15] A. J. Ramirez and B. H. C. Cheng, "Design patterns for developing dynamically adaptive systems," pp. 49–58, 2010.

[16] G. Nitschke, M. Schut, and A. Eiben, "Emergent specialization in biologically inspired collective behavior systems," in *Intelligent Complex Adaptive Systems*, A. Yang and Y. Shan, Eds. IGI Publishing, 2008, pp. 215–253.

[17] 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.

[18] P. Kersch, R. Szabo, Z. Kis, M. Erdei, and B. Kovács, "Self organizing ambient control space: an ambient network architecture for dynamic network interconnection," in *Proc. of 1st ACM Wkshp. on Dynamic Interconnection of Networks*. ACM, 2005, pp. 17–21.

[19] J. L. Fernandez-Marquez, J. L. Arcos, and G. D. M. Serugendo, "A decentralized approach for detecting dynamically changing diffuse event sources in noisy WSN environments," *Applied Artificial Int.*, vol. 26, no. 4, pp. 376–397, 2012.

[20] R. Nagpal, "Programmable self-assembly using biologically-inspired multiagent control," in *1st Int'l. Joint Conf. on Autonomous Agents and Multiagent Systems: Part 1*, 2002, pp. 418–425.

[21] S. Selvakennedy, S. Sinnappan, and Y. Shang, "A biologically-inspired clustering protocol for wireless sensor networks," *Computer Communications*, vol. 30, no. 14-15, pp. 2786–2801, 2007.

[22] J. Sudeikat and W. Renz, "Toward systemic mas development: Enforcing decentralized self–organization by composition and refinement of archetype dynamics," *Engineering Environment-Mediated Multi-Agent Systems*, pp. 39–57, 2008.

[23] M. Casadei, L. Gardelli, and M. Viroli, "Simulating emergent properties of coordination in maude: the collective sort case," *Electronic Notes in Theoretical Computer Science*, vol. 175, no. 2, pp. 59–80, 2007.

[24] L. Gardelli, M. Viroli, M. Casadei, and A. Omicini, "Designing self-organising MAS environments: the collective sort case," *Env.s for Multi-Agent Systems III*, pp. 254–271, 2007.

[25] M. Casadei, M. Viroli, and L. Gardelli, "On the collective sort problem for distributed tuple spaces," *Science of Computer Programming*, vol. 74, no. 9, pp. 702–722, 2009.

[26] J. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien, "The dynamics of collective sorting robot-like ants and ant-like robots," in *Proc. of 1st Int'l Conf on Simulation of Adaptive Behavior on From Animals to Animats*, 1991, pp. 356–363.

[27] T. Wang and H. Zhang, "Collective sorting with multiple robots," in *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE Int'l Conf. on*. IEEE, 2004, pp. 716–720.

[28] H. Mühleisen, A. Augustin, T. Walther, M. Harasic, K. Teymourian, and R. Tolksdorf, "A self-organized semantic storage service," in *Proc. of the 12th Int'l Conf. on Info. Integration and Web-based App. & Serv.* ACM, 2010, pp. 357–364.