Designing Self-Organizing Software with a Design Pattern Catalog: A Case Study

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
Centre Universitaire d'Informatique
Carouge, SWITZERLAND

***Abstract*—Large and ultra-large scale software systems require substantial self-adaptation capabilities. Investigations of self-organizing mechanisms, often inspired by phenomena occurring in natural or societal complex adaptive systems, have produced an array of self-adaptation techniques that are eminently scalable, since they build emergent and stable global behaviors based only on simple local rules.**

However, the principled design of self-adaptive software using self-organizing mechanisms in a modular and reusable way is 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 such design patterns to the modeling of an existing, complex self-organizing system: the Mycoload clustering and load balancing application for unstructured peer-to-peer networks.

By re-engineering the design of Mycoload via a set of bio-inspired patterns, we show how the different intertwined mechanisms participating in the self-organizing dynamics of the application can be isolated, and how their relationships can be clarified. This exercise also allowed us to identify another self-organization design pattern, SPECIALIZATION, which we present here for the first time, and to identify other functional mechanisms of Mycoload which may be captured in the future as yet other patterns.

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

I. INTRODUCTION

As modern computing and communications environments continue to expand in scale and complexity, we are witnessing the emergence of what are sometimes called Ultra-Large Scale Systems [1]. These systems are characterized by huge numbers of components (ranging from several thousands, up to millions of servers, laptops, mobile phones or PDAs), heterogenous properties and ownership of those computing devices, high levels of dynamism, and indistinct system boundaries. They enable and host a range of diverse applications: examples include crowd steering systems, recommender systems, social networks, wireless sensor networks, and peer-to-peer systems.

The development of these applications faces significant challenges, such as extreme scalability, real-time response, and failure tolerance. Because of the highly dynamic environments in which they evolve, such systems must be self-adaptive, but using centralized "command-and-control" designs for deployment, operation, or management quickly becomes impractical at these large scales. Therefore, researchers have been looking at the mechanisms of self-organization in complex adaptive systems, in order to draw inspiration and experiment with decentralized self-adaptation techniques. Self-organizing systems, for example in biology, have several appealing characteristics: they are robust, resilient, able to adapt to environmental changes and able to achieve complex behavior using a simple set of local rules [2].

However, in order to be useful for the engineering of self-adaptive systems, these self-organizing mechanisms must be made available to software designers in a principled way. To address this problem, several attempts have been made in the last few years to present them under the form of design patterns [3], [4]. In previous works [5], we proposed a catalog of bio-inspired self-organizing design patterns. In that catalog, we defined relations between the identified patterns, for instance, how basic patterns take part in composed, higher-level patterns. The motivation of this work was, at the same time, to organize the growing body of knowledge on self-organizing mechanisms, and enable the design and implementation of self-organizing applications in a modular and reusable way.

In this paper, we focus now on the use of these patterns to model a specific application. We leverage the patterns in our catalog to show how they can be used to express the design of Mycoload [6], a self-organizing system for clustering and load-balancing in unstructured peer-to-peer networks, and how that re-engineering effort is useful in highlighting and isolating other reusable self-organizing mechanisms. Thus, our contributions are as follows: (1) we show how to model, by means of these design patterns, a complex self-adaptive system (such as Mycoload) that uses multiple self-organizing mechanisms; (2) we isolate and specify a new design pattern: the SPECIALIZATION pattern; and (3) we identify clearly a set of patterns that contribute to the adaptive dynamics of Mycoload. As a consequence of this pattern-based modeling we can better reason about Mycoload in order to support the maintenance of the system and its already planned extensions; this also paves the way to re-engineering Mycoload within the framework of a middleware such as BIO-CORE [7], where the responsibilities of application-level elements are clearly separated from those of the middleware

(i.e., basic self-organization mechanisms can be executed by the middleware itself).

This paper is structured as follows: Section II discusses related work on self-organizing mechanisms and design patterns. Section III briefly provides background on a set of basic and composed mechanisms that are used later on in the modeling of Mycoload. Section IV explains the self-organized dynamics of Mycoload. Section V describes the new SPECIALIZATION, and Section VI discusses how the Mycoload design is expressed in terms of SPECIALIZATION and other patterns. We conclude in Section VII with a discussion and an outline of our future work.

II. RELATED WORK

Researchers have repeatedly taken inspiration from biological systems, and have identified self-organizing mechanisms that can be mimicked in computing systems. The allows results that go beyond traditional approaches [8], [9]. However, these self-organizing mechanisms have been often applied in an ad hoc manner, with varying interpretations and no well-defined distinction or boundaries.

Works on self-organizing design patterns [10], [3], [4], [11] have tried to organize this field, and to pave the way for the application of self-organizing mechanisms in a modular and reusable way. These self-organizing design patterns provide a common solution for solving recurrent problems in the engineering of self-organizing systems, specifying when, where and how each mechanism should be used. The majority of works in this area have focused on the codification of one or more patterns, and a number of them have been proposed in the last few years. The most recent works some works have attempted to augment the level of support, e.g. by proposing a catalog of patterns that highlight how they relate to each other [5], or by providing concrete implementation descriptions [12]. However there is still a lack of guidance on how to use identified self-organized patterns in the engineering of self-adaptive systems.

Our work advances in that direction, since it may be the first that tries to 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 most similar to the work by Ramirez et al. [13], who analyzed a list of adaptation design patterns and re-engineered an application using those patterns, and presented several advantages of using such design patterns compared to a framework-oriented approach. Since Ramirez et al. focused on adaptation design patterns, we consider that a similar effort must be done also for evaluating and promoting self-organizing design patterns.

III. SELF-ORGANIZATION DESIGN PATTERNS

In previous works [5], we presented a list of self-organization design patterns (which have largely been derived from biologically inspired metaphors) analyzing the relations between them. One major contribution of this work—together with a detailed codification of known patterns—was a three-layer classification (basic, composed and high-level patterns), shown in Figure 1. This classification also defined composition/usage relations among the corresponding self-organization mechanisms: 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. Our work ([5]) shows how a limited number of these basic mechanisms are at the basis of a large set of powerful self-organizing mechanisms that have been experimented with in the literature, such as gossip, gradients, and morphogenesis, and how they can be themselves be expressed as design patterns. Our classification scheme thus offers a better understanding of complex as well as basic self-organizing mechanisms, and shows how these can be modeled in a modular way. Most importantly, it also enables the design of self-organizing applications in terms of these modular and reusable building blocks.

In this section, we briefly review a selected set of the patterns previously proposed in [5] by giving only the problem they address and the solution they provide. These will become useful for the discussion of the analysis of the Mycoload system, found in Section VI.

1) **SPREADING***:* The SPREADING pattern is a basic pattern for information dissemination or diffusion, aiming at incrementing the global knowledge of the system by using only local interactions.

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

Solution. A copy of information received or held by an agent is sent to neighbors and propagated over the network. Information spreads progressively through the system and reduces the agents' lack of knowledge while keeping the constraint of local interaction.

The SPREADING mechanism has been applied to wide number of applications: a few include swarm motion coordination [14], coordination in games [15], problem optimization [16], but this mechanism comprises a significant part of the dynamics of many self-organizing systems.

2) **AGGREGATION***:* The AGGREGATION pattern is a basic pattern for information fusion, aiming to reduce excess information while retaining enough meaningful information to enable local decision-making.

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

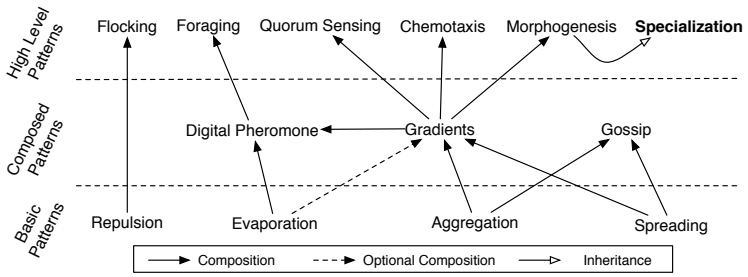

Figure 1. Self-Organization Design Patterns

information.

Solution. Aggregation consists in locally applying a fusion operator to process and synthesize information. Fusion operators are classified into four different groups: filters, transformers, mergers and aggregators [17].

Aggregation has been used, for example, in the ACO algorithm [18] for the aggregation of pheromones, simulating the higher concentrations that result when two or more pheromones are close to each other. Aggregation is also frequently used together with digital pheromones, as with the autonomous coordination of swarming UAVs in [14].

3) **EVAPORATION***:* The EVAPORATION pattern is a basic pattern that helps to deal with dynamic environments where information used by agents can become outdated, or the freshness of information is crucial for the agents' decisions.

Problem. Outdated information cannot be directly 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 progressively reduces the relevance of information. Thus, recent information can be easily determined to be more relevant than information processed some time ago.

Evaporation has been used in Dynamic Optimization, among other areas. Examples of algorithms using evaporation are ACO [18] and Quantum Swarm Optimization Evaporation (QSOE) [19]. In some cases, evaporation is facilitated by associating a freshness parameter with the information [20].

4) **GOSSIP***:* The goal of the GOSSIP pattern is to obtain an agreement about the value of some parameters in the system in a decentralized way. Agents in the system collaborate to progressively reach this agreement. Each contributes by aggregating its own knowledge with its neighbors' knowledge and by spreading this aggregated knowledge. Thus, GOSSIP is composed of the SPREADING and AGGREGATION patterns.

Problem. In large-scale systems, agents need to reach an agreement across all agents with only local perception in a decentralized way.

Solution. Information spreads to neighbors where it is aggregated with local information. Aggregates are spread further and their value progressively improves the agreement.

Kempe et al. [21] analyze a simple gossip-based protocol for the computation of sums, averages, random samples, quantiles, and other aggregate functions. Gossip has also been used, *e.g.*, for coordination in large convention spaces [22] using aggregation based on evolutionary algorithms.

5) **GRADIENT***:* The GRADIENT pattern is an extension of the SPREADING pattern where the information is propagated in such a way that it provides 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 values indicating the sender is closer, as in ants pheromone models. Additionally, the GRADIENT pattern uses AGGREGATION to merge gradients created by different agents or to merge gradients coming from the same agent but through different paths.

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 direct communication range

Solution. Information spreads from the location where it is initially created or 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 (as when information that is further away is modeled using lower concentrations relative to closely-situated information).

The GRADIENT pattern has been used, among other applications, for the coordination of swarms of robots [14], coordination of agents in video games [15], and routing in ad-hoc networks [23].

IV. MYCOLOAD

Mycoload [6] is a self-organizing system for clustering and load-balancing in unstructured peer-to-peer networks.

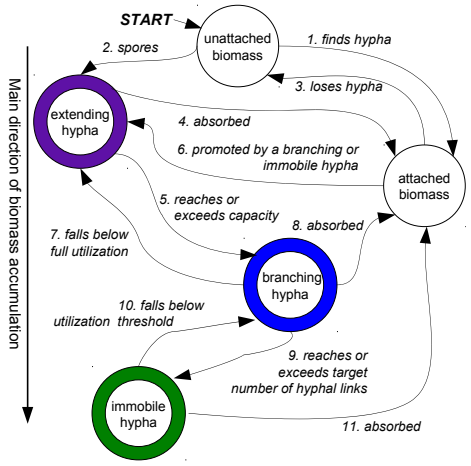

Figure 2. Mycoload protocol states and transitions.

Mycoload is an extension to Myconet [24], a protocol for constructing superpeer-based overlays, inspired by the growth patterns of fungal root systems, called *hyphae*.

In this section, we provide some background on the characteristics and behavior of Mycoload in order to introduce and motivate the SPECIALIZATION pattern presented in Section V, and to inform the in-depth discussion of its design in the light of the self-organization design patterns found in Section VI. As detailed description of protocol rules or results for Myconet and Mycoload are out of the scope of this paper, readers are referred to [24] for a more extensive discussion of Myconet's topology management and to [6] for a comparison to other self-organizing load-balancing approaches.

Mycoload is built as an application on the Myconet overlay protocol. Myconet uses a fungal metaphor in which superpeers are considered to be hyphae, and ordinary, non-superpeer nodes are considered be be biomass*biomass* (that is, nutrients) to be moved among the hyphae as needed. Mycoload nodes use an abstract capacity measure to capture the heterogenous properties of nodes, with higher capacities representing an higher suitability to act as a superpeer and provide services to other peers. (The terms *peer* and *node* are used interchangably in this paper.)

Myconet constructs and maintains a strongly interconnected overlay that quickly converges to an optimal number of superpeers. 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 mantaining the overlay, and peers are promoted or demoted to those states based on how well they are able carry out those responsibilities. The states and transitions are shown in the diagram in Figure 2. When promoted from biomass to the first, *extending* hyphal state, superpeers are mainly focused on exploration, acting as connection points for new biomass peers entering the network. Peers will remain in this state until connected to at least one other node providing the same service type. *Branching* hyphae are superpeers that have reached their target level of utilization (i.e., they have connected to sufficient biomass peers to reach a level of utilization proportional to their capacity attribute); they manage the number of extending superpeers in the network while growing new hyphal interconnections. Finally, *immobile* hyphae have reached levels of full biomass connection and a target number of hyphal interconnections, and are considered to be relatively stable, having remained in the network long enough to transition through all the previous protocol states.

When superpeers contact each other, peers of lower protocol state and lower capacity transfer (part of) their attached biomass peers towards higher-level superpeers in order to saturate the capacity of the latter. They also act as matchmakers for connected peers offering different service types, enabling cluster formation. Should a hyphal peer for any reason fall below a utilization threshold, it may demote itself to a lower protocol state, possibly reverting to become an ordinary, biomass peer.

Mycoload extends the Myconet protocol outlined above with rules that aim to maintain clusters of peers offering the same service type. Self-organized load balancing is then applied to the peers within each cluster (following a variant of the algorithm in [25]) to distribute requests for that services of that type in an efficient way among the peers' queues. Load balancing is performed between same service-type neighbors within a cluster, with local exchanges made to achieve the goal of queue lengths proportional to the capacity of each peer. When a load-balancing operation is performed, a peer selects a random same-type superpeer neighbor; as biomass peers have only a single neighbor, they will be the ones to initiate the load-balancing operations with their parent. This way, load-balancing operations tend to occur preferentially between the more-capable superpeers. This helps ensure that jobs will be balanced throughout the cluster. "Ideal" queue lengths are calculated by determining the total number of jobs in both queues and dividing the jobs proportionally based on p_a and p_b's capacities. If the queues are unbalanced, jobs are transferred from the queue that is over its ideal length to the queue that is under its ideal.

Mycoload's approach is related to other topology construction and management protocols for unstructured peer-to-peer networks such as SG-1 [26] and T-MAN [27]. SG-1 uses a two-level model (peers and superpeers), while Myconet's multi-level approach adds additional mechanisms for self-healing and self-optimization. T-MAN uses a ranking function to determine relative distances of a peer to other peers in its local neighborhood, which enables clustering but does not address load-balancing efficiency. Mycoload

improves load balancing by using intra-cluster hierarchies to position more capable nodes centrally within each cluster.

In its current implementation, each Mycoload peer offers only a single service type, and each clustered grouping collects peers of that type. The process of designing of an extension to Mycoload where peers can host multiple types of services is the original motivation for modeling the current application as a set of patterns that tease out the different elements of its self-organizing dynamics. That pattern-based analysis is presented in Section VI, and yielded the identification of another design pattern for self-organization, SPECIALIZATION, which we discuss in the next section.

V. SPECIALIZATION PATTERN

In this section we introduce a new self-organization design pattern called SPECIALIZATION that will be used in the next section in the design of the Mycoload system. Specialization is a mechanism widely used in complex system for achieving improved efficiency by exploiting the natural heterogeneity of the entities (e.g. computational power, memory available, sensors, actuators, battery level or location). In the SPECIALIZATION pattern, individual entities are assigned a specific role depending on their capabilities and contextual local information. This assignment can be be considered as "open" specialization when the set of possible roles is not known a priori, or "closed" specialization otherwise. A useful survey of specialization in self-organizing systems can be found in [28].

In SPECIALIZATION, entities participating in the system will change the rules under which they operate depending on features or properties of the entity itself or contextual information from its environment or its 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, or a node with a large amount of bandwidth connections could elect to act as a router for traffic transmitted by other nodes. These roles may further need to adapt dynamically to changing system conditions and requirements.

This kind of specialization is frequently seen in everyday life. People are normally specialized to perform certain tasks for they are more capable, though this specialization does not prevent them from executing other tasks when necessary.

The SPECIALIZATION pattern is presented following the self-organization design pattern structure proposed in [5].

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 people in a social environment.

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 [5] 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 [5], 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).

$$\begin{aligned} &\texttt{state\_evolution} :: \langle L, [cInf, State, C]\rangle \xrightarrow{r_{sp}} \\ &\langle L, [cInf, State', C]\rangle \\ &\qquad \textit{where} \quad State' = \pi(cInf, State, C) \end{aligned} \tag{1}$$

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.

Known Uses: Specialization has been used by a large number of self-organizing applications. Examples include: (1) In the Quantum Swarm Optimization Evaporation (QSOE) [19] extension of the Particle Swarm Optimization algorithm, where particles are divided into "explorer" and "exploiter" roles. Explorers are those that seek new optima (and hence do not aim to converge), while exploiter particles are those that collaborate to increment the accuracy of a known solution (and do aim to converge on the optimum solution). (2) Overlay networks where some nodes decide to become routers based on their available resources and their connectivity with other nearby nodes [29] (3) 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 [30]. (4) To balance the load among nodes with different services, Mycoload [6], builds a superpeer topology where more powerful nodes adopt several different, specialized roles in the creation and maintainance of the overlay. (5) [31] 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 position with some of the node, typically communicated via a GRADIENT. Thus, the MOPHOGENESIS pattern is of the same family but more specific than SPECIALIZATION.

VI. DESIGN OF MYCOLOAD WITH NEW AND EXISTING PATTERNS.

In this section, we analyze an existing self-organizing system, Mycoload, and describe its self-adaptive behavior in terms of the catalog of design patterns discussed in Section III. The overall behavior of Mycoload emerges as the result of the combined action of multiple self-organizing mechanisms. These were implemented on the basis of a biological metaphor applied to peer-to-peer overlays, and underwent further incremental refinement and tuning over time. We approached this case study as a redesign project, with the goal of identifing, isolating, and modularizing the various self-organizing mechanisms that contribute to Mycoload's dynamics.

The primary goal of Mycoload is to *maximize throughput by balancing load queues for multiple services across many peers in a distributed service network with heterogeneous capabilities.* We decomposed this goal into three primary features that the system must have: First, the system must (*A*) *construct an efficient overlay network connecting the peers to each other in such a way as to facilitate load balancing*. Second, it should (*B*) *collect peers offering similar services into connected subgroups so that they can interchange jobs with other peers in the same subgroup*. Third, it should (*C*) *shift jobs between peers such that peers with higher capacity are given proportionally more work*. The relationship between these features is shown as a UML use case diagram in Figure 3.

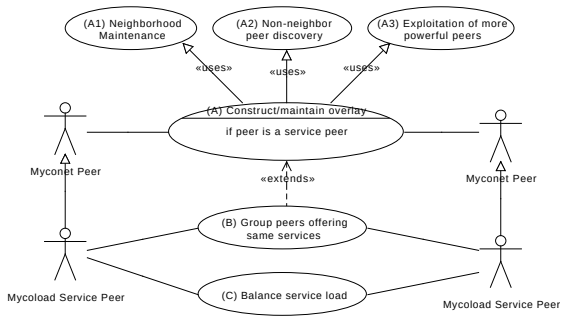

Figure 3. Relation between self-organizing mechanisms in Mycoload

A. Overlay Construction

Feature (*A*), the construction of the overlay network, requires a number of subfeatures that we discuss in detail below: (*A1*) *peers must maintain small neighborhoods for local interactions*; (*A2*) *peers need to be able to discover appropriate neighbors when entering the system and maintaining the overlay*; and (*A3*) *the overlay must exploit the heterogeneity of the system by giving more powerful peers additional roles in the maintenance of the overlay*.

(A1) Neighborhood maintenance: Peers need to have information about other peers in their neighborhoods and those peers' capabilities, and they must keep this information up-to-date. In Mycoload this is partially achieved by applying the SPREADING pattern. To implement SPREADING, peers periodically update neighbor peers with their own status, including information such as their protocol state, their capacity, and their number of neighbors. This information could be further relayed, if needed, though in Mycoload it does not need to pass further than the second hop (that is, to a neighbor's neighbor). In order to ensure the information about neighbor peers remains current, the exchanged status is marked with timestamps, and expires if not refreshed by the subsequent spreading of updated information. This is similar to the EVAPORATION pattern, but is simpler since the importance of the information is not progressively reduced as it gets older but remains within a period of validity. The mechanisms which take part in the neighborhood maintenance feature of Mycoload are shown in Figure 4.

(A2) Non-neighbor discovery: Peers need to be able to discover appropriate neighbors when entering the system and when building and maintaining the overlay. For this, the ability to select a random peer from elsewhere in the network is highly desirable. At large scales, since the overlay

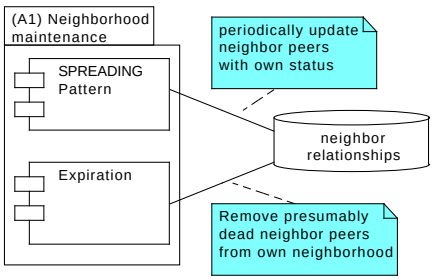

Figure 4. Self-organizing overlay neighborhood maintenance mechanisms in Mycoload

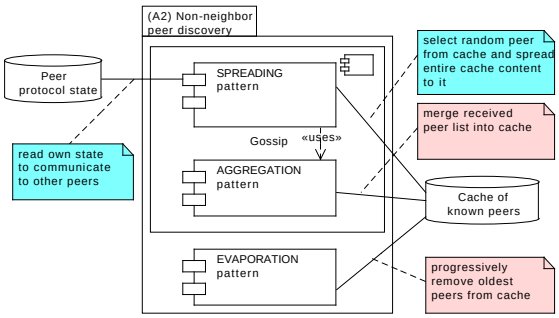

Figure 5. Self-organizing mechanisms for maintaining a view of non-neighbor peers in Mycoload

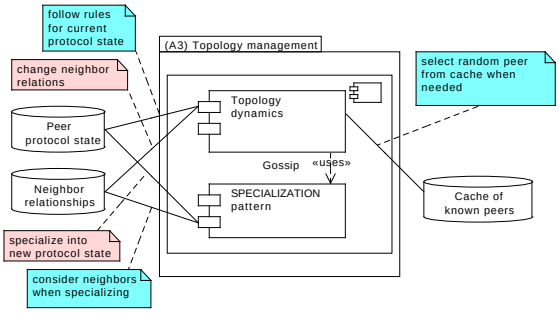

Figure 6. Self-organizing topology management mechanisms in Mycoload

membership may be highly dynamic and there are no central authorities maintaining complete membership lists, peers need to achieve this through a mechanism that works locally while maintaining a limited set of data.

One way to achieve this is through the GOSSIP pattern, enabling nodes to share their knowledge about other active nodes outside of the context of overlay neighbor relationships. With each interaction, pairs of peers contribute to a joint agreement on which nodes are likely to be fresh, and this information will then be spread to other nodes through subsequent exchanges. The GOSSIP mechanism is thus composed of two sub-patterns, SPREADING and AGGREGATION. Both of these mechanisms, in this case, operate on a list of known peers that is maintained at each local node. Each peer periodically selects a known peer and sends it a copy of its entire list (SPREADING). Upon receiving such a message, the recipient combines that list with its own (AGGREGATION), along with an entry for the sending peer timestamped with the current time.

In order to prevent the lists from growing indefinitely (and to eventually remove peers that are no longer partipating in the gossip exchanges), peers are removed from the list via an EVAPORATION mechanism. Entries with the oldest time are progressively discarded, keeping the list at a fixed, relatively short length. All of the above patterns are implemented by the Newscast protocol [32], which Mycoload uses as an off-the-shelf component to maintain these lists as a source of random nodes from elsewhere in the network when needed. The relations between these patterns are shown in Figure 5.

(A3) Exploit more powerful peers: Mycoload's design is intended to deal with systems of peers that may have greatly differing capabilities and resources (that is, some peers are much more powerful than others). The presence of this kind of heterogeneity brings about an opportunity for improved performance, by having a limited number of peers perform certain advanced functions, and is a clear pointer that the SPECIALIZATION pattern should be considered (this is the main principle underlying superpeer overlays in general).

To implement this, each peer considers its own status and the situation in its neighborhood (primarily, its number of neighbors relative to an ideal capacity, a measure based on its available resources, and its own capacity compared to that of its neighbors). Each peer uses this local information to determine whether to adopt a new role (as a superpeer in one of the hyphal stages described in section IV), and switching the set of protocol rules it uses to manage neighbor relationships accordingly.

The information considered by a peer when making the decision on whether to adopt a new, specialized state may also include messages from other peers (in this case, superpeers with a broader view of the network); these are received "hints" that the peer is well-positioned to be promoted and serve as a new superpeer, or that it has now become redundant, and it can best reduce inefficiency by demoting itself.

The full topology maintenance rules followed in each specialized state cannot be described in detail here due to space limitations; a detailed description can be found in [6]. These multiple sets of rules are captured in Figure 6 as "topology dynamics".

(B) Collect peers offering similar services into groups

In the distributed service network scenario targeted by Mycoload, peers may offer different services and jobs of

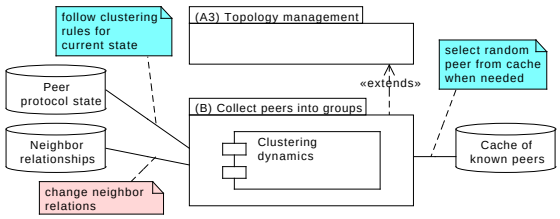

Figure 7. Self-organizing clustering mechanisms in Mycoload

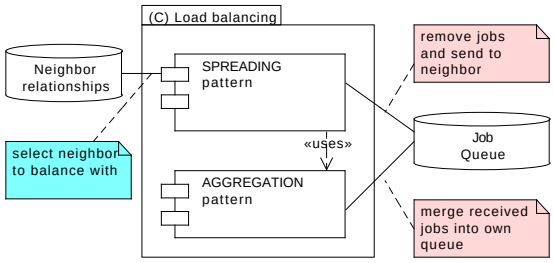

Figure 8. Self-organizing load-balancing mechanisms in Mycoload

a certain type may only be processed by peers offering a service of the same type. In order to exchange and load-balance jobs, peers of similar type need to be able to find each other. This is achieved by *clustering* same-type peers into neighborhoods served by one or more hyphal peers of the same service type, thereby adjusting the topology of the overlay to build connected groups of nodes hosting the same type of service while continuing to exploit the specializations resulting from each node's heterogeneous capabilities.

The basic purpose of the clustering dynamics in Mycoload is thus to extend the basic topology dynamics that build the superpeer overlay (*A3*) with considerations of neighbor types. Peers that have not managed to achieve a target number of same-type neighbors explore their environment by querying attached superpeers to find if they have same-type neighbors; the superpeers also act as matchmakers, attempting to connect their own neighbors with similar-type peers at another step further removed. If peers are unable to find matches using these mechanisms, they also engage in random exploration using the the peer list maintained by Feature (*A2*). These relationships are shown in Figure 7.

During the modeling of Mycoload, after the mechanisms described in section VI-A had been isolated, it became clear that the self-organized clustering approach can itself be captured into a self-contained unit of behaviors separate from the other mechanisms. We have not yet formalized self-organized clustering into its own pattern, but this is a direction for immediate future work.

(C) Load Balancing

Since all the responsibility for building the overlay is handled by features (*A*) and (*B*), self-organizing load balancing can then rely on the properties of the overlay to perform its work. Effectively, load-balancing is a problem of equalizing levels of work among peers offering the same type of service, scaled according to each peer's capacity.

This is a natural match for the GRADIENT pattern/ By considering the jobs in a peer's queue as analogous to the quantity of some chemical marker and the capacity of a peer as a volume in which the chemical is present, a peer can determine the relative concentration of the chemical marker relative to its neighbors, and, based on this information, determine whether jobs need to be transferred in order to achieve a balanced load. Thus, excess jobs diffuse through the system, with more jobs accumulating at higher-capacity peers.

GRADIENTS are implemented via two more basic patterns: SPREADING and AGGREGATION. For the SPREADING portion, peers have access to information about their direct neighbors via the overlay network, including their capacity and queue length. Peers periodically randomly select a neighbor with which to balance queues, and the peer with the higher relative concentration transfers the job to the other. AGGREGATION is performed when a peer receives jobs from a neighbor and they are merged into its own queue, ordering them such that each receives fair service based on its time of entry and other features. The relationship between these patterns is shown in Figure 8.

VII. DISCUSSION AND CONCLUSIONS

We have presented a case study showing the pattern-based modeling of an entire self-organized software system, Mycoload. This is something that, to our knowledge, has not yet been documented elsewhere in the literature about the design of self-organized software. Most research efforts have resulted in the identification and extraction of self-organization design patterns (as discussed in Section II), typically focusing on a single pattern that characterizes the primary dynamic of a particular system. This paper addresses the next stage of this problem: identifying patterns within an existing, complex distributed system that relies on multiple self-organizing mechanisms, and using those patterns as the basis for a modularized redesign of the system. We see this is a significant step toward the development of a methodology for engineering self-organizing software systems.

The case study also demonstrates the effectiveness of a well-organized catalog of design patterns [5] in modeling the design of an existing self-organizing software system. Many of the patterns of self-organization identified in Mycoload are at the basic level, and as such are ideal candidates for reuse. One possible approach is to re-engineer a system like Mycoload on top of a distributed infrastructure or middleware (such as TOTA [33] or BIO-CORE [7]) that

itself provides these features, allowing the application to focus on higher-level self-organizing dynamics and domain-specific concerns.

A continuing challenge in the design and implementation of self-organizing systems is to understand the global emergent behaviors and properties that result from rules followed by lower-level components and their self-organizing interactions. The ability to identify patterns and their role within the context of the system at large provides leverage for this problem: each pattern represents a well-understood piece of dynamics with its own micro-/macro-level linkages; thus, focus can be shifted to the higher-level dynamics that result from the composition of the identified lower-level mechanisms, and the effects of adjusting their rules and parameters. Since in systems with emergent behavior simple changes can result in large modification of the overall dynamics, the modularization of the system also improves the level of support for the maintenance and evolution of the system over time, as changes can be more easily targeted to contained areas of functionality.

The benefits described above will be reflected in our immediate future work, since we intend to leverage the pattern-based design model to develop a version of Mycoload that can handle the clustering and load-balancing of multi-service peers, and to explore the possibility of using a self-organizing middleware as a substrate for this implementation.

Beyond demonstrating the usefulness of a catalog of self-organization design patterns, another benefit of the design analysis carried out in Section VI was the isolation of other self-organization mechanisms that are integral to its overall behavior: (1) the SPECIALIZATION pattern (as described in Section V), which captures a mechanism that is present in a number of other self-organized systems and solves the recurring problem of self-selected differentiation of roles and responsibilities among (possibly heterogeneous) system elements or agents; and (2) the clustering mechanism that enables collection of entities into similar-type groupings within the context of a collection of system elements. Clustering is also a recurring problem in self-organizing systems, where elements often need to be assorted into homogeneous groupings according to some property of the entities or requirement of the system. Several self-organizing algorithms for clustering have been developed, for example, those based on biological phenomena, such as brood sorting and cemetery formation by social insects and other swarm approaches [34], [35]. Capturing and abstracting the self-organizing dynamics of clustering as a design pattern is also a likely direction for our future work.

ACKNOWLEDGEMENT

The authors would like to give special thanks to Daniel J. Dubois, Elisabetta Di Nitto, and Nicolò M. Calcavecchia for their contributions to the development of Mycoload and to Rachel Greenstadt for her contributions to the development of Myconet.

REFERENCES

[1] L. Northrop, P. Feiler, R. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan *et al.*, "Ultra-large-scale systems: The software challenge of the future," *Software Engineering Institute*, 2006.

[2] F. Dressler and O. B. Akan, "A survey on bio-inspired networking," *Computer Networks*, vol. 54, no. 6, pp. 881–900, 2010.

[3] L. Gardelli, M. Viroli, and A. Omicini, "Design patterns for self-organizing multiagent systems," in *2nd International Workshop on Engineering Emergence in Decentralised Autonomic System (EEDAS) 2007*, T. D. Wolf, F. Saffre, and R. Anthony, Eds. ICAC 2007, Jacksonville, Florida, USA: CMS Press, University of Greenwich, London, UK, June 2007, pp. 62–71.

[4] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," in *Proceedings of the 4th international conference on Engineering self-organising systems*, ser. ESOA'06. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 28–49. [Online]. Available: http://portal.acm.org/citation.cfm?id=1763581.1763585

[5] 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 Journal (to appear)*, 2012.

[6] 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. [Online]. Available: http://dblp.uni-trier.de/db/conf/saso/saso2011.html#ValettoSDNC11

[7] J. L. Fernandez-Marquez, G. Di Marzo Serurendo, and S. Montagna, "Bio-core: Bio-inspired self-organising mechanisms core. (to appear)," in *6th International ICST Conference on Bio-Inspired Models of Network, Information, and Computing Systems (Submitted)*, 2011.

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

[9] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli, "Case studies for self-organization in computer science," *Journal of Systems Architecture*, vol. 52, pp. 443–460, August 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1163824.1163826

[10] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85029-8_4

[11] 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.

[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. [Online]. Available: http://doi.acm.org/10.1145/2095050.2095083

[13] A. J. Ramirez and B. H. C. Cheng, "Design patterns for developing dynamically adaptive systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '10. New York, NY, USA: ACM, 2010, pp. 49–58. [Online]. Available: http://dx.doi.org/10.1145/1808984.1808990

[14] H. V. D. Parunak, M. Purcell, and R. O'Connell, "Digital pheromones for autonomous coordination of swarming uavs," in *The First AIAA Unmanned Aerospace Vehivales, Systems, Technologies, and Operations*, 2002, pp. 1 – 9.

[15] M. Mamei and F. Zambonelli, "Field-based motion coordination in quake 3 arena," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, ser. AAMAS '04. IEEE Computer Society, 2004, pp. 1532–1533.

[16] C. Blum, "Beam-aco: hybridizing ant colony optimization with beam search: an application to open shop scheduling," *Computers & Operations Research*, vol. 32, no. 6, pp. 1565–1591, 2005.

[17] G. Chen and D. Kotz, "Solar: A pervasive-computing infrastructure for context-aware mobile applications," Department of Computer Science, Dartmouth College Hanover, NH, USA 03755, Tech. Rep., 2002.

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

[19] J. L. Fernandez-Marquez and J. L. Arcos, "An evaporation mechanism for dynamic and noisy multimodal optimization," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ser. GECCO '09. ACM, 2009, pp. 17–24. [Online]. Available: http://doi.acm.org/10.1145/1569901.1569905

[20] D. Weyns, N. Boucké, and T. Holvoet, "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*. New York, NY, USA: ACM, 2006, pp. 842–849.

[21] 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.

[22] 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.

[23] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, ser. WMCSA '99. IEEE-CS, 1999.

[24] 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.

[25] E. Di Nitto, D. J. Dubois, R. Mirandola, F. Saffre, and R. Tateson, "Applying self-aggregation to load balancing: experimental results," in *BIONETICS'08*. ICST, 2008, pp. 14:1–14:8.

[26] A. Montresor, "A robust protocol for building superpeer overlay topologies," in *P2P'04*. Zurich, Switzerland: IEEE, Aug. 2004, pp. 202–209.

[27] M. Jelasity and O. Babaoglu, "T-Man: Fast gossip-based construction of large-scale overlay topologies," *University of Bologna, UBLCS-2004-7, Italy*, 2004.

[28] 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.

[29] 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 *Proceedings of the 1st ACM workshop on Dynamic interconnection of networks*. ACM, 2005, pp. 17–21.

[30] 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 Intelligence Journal*, 2012 (to appear).

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

[32] M. Jelasity, W. Kowalczyk, and M. Van Steen, "Newscast computing," Vrije Universiteit Amsterdam Department of Computer Science Internal Report IR-CS-006, 2003.

[33] M. Mamei and F. Zambonelli, "Spatial computing: The TOTA approach," in *Self-star Properties in Complex Information Systems*, ser. Lecture Notes in Computer Science, vol. 3460/2005. Springer Berlin / Heidelberg, 2005, pp. 307–324. [Online]. Available: http://www.springerlink.com/content/h8ce567211djhqhk/

[34] S. Zhongzhi *et al.*, "A clustering algorithm based on swarm intelligence," in *Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on*, vol. 3. IEEE, 2001, pp. 58–66.

[35] 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.