Self-Organizing Replica Placement – A Case Study on Emergence

Klaus Herrmann
University of Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätstr. 38, D-70569 Stuttgart
klaus.herrmann@acm.org

Abstract

The concept of self-organization is rapidly gaining importance in the area of distributed computing systems. However, we still lack the necessary means for engineering such system in a standardized way since their common properties are rather abstract, and the mechanisms from which self-organization emerges are too diverse. Therefore, it has become common practice to engineer computing systems by taking inspirations from well-known case studies of biological systems. However, the concepts found in such systems are in many cases only partially transferable to the domain of distributed computing systems since biological systems are subject to vastly different constraints compared to those in a computing system. Our contributions in this paper are the following: (i) *We present a case study of a self-organizing software system that originates from the domain of distributed computing systems. Therefore, its concepts can be exploited in other distributed computing systems much more directly.* (ii) *We give a detailed analysis of the emergent properties of the system and the mechanisms by which they arise.* (iii) *We generalize the mechanisms by which self-organization emerges in this system and present a catalog of design questions that may help engineers in creating arbitrary self-organizing systems.*

1 Introduction

In recent years, the research area revolving around *self-organization* and *emergence* in computer science has grown up from a niche discipline to a major topic. This is especially true in the area of distributed systems where the following factors contribute to this development:

1. Emerging technologies like wireless sensor networks [1] and mobile ad hoc networks [12] result in highly dynamic distributed systems that do not provide any clear-cut management interfaces. Such systems tend to collate spontaneously such that no single administrative authority can assume responsibility for their organization or management.

2. Large-scale distributed information systems like Content Delivery Networks [34], Peer-to-Peer systems [2], and Publish/Subscribe infrastructures [9] escape classical management attempts due to their sheer size. Dynamic usage patterns of clients lead to the inherent need for timely adaptation. However, centralized management is prohibitive as this would reduce scalability drastically.

3. The growing dynamics of a large variety of distributed computing systems raises the need for more autonomous systems that may adapt to changing working conditions with minimal manual intervention. In an industrial context, this leads to a reduction of management costs and to a more timely adaptation of the respective systems.

Case studies of self-organizing systems outside the computer science domain (e.g. from biology) show that a self-organizing system commonly consists of a group of relatively simple entities (e.g. ants, bees, or fish) that interact to achieve complex adaptive group behavior. From a technical viewpoint, lower-level components that are simple and homogeneous are easy to manufacture and maintain. Moreover, the redundancy inherent to such a system implies robustness as well as scalability. But how can the lower-level components of a system and their interactions be designed to produce the desired behavior of the overall technical system? Despite several years of intense research, the field of *self-organizing software systems* is still in its infancy, and we lack the necessary general means for developing self-organizing software systems in a reliable and dependable way.

Studies of individual biological systems have enabled technical solutions for specific problems. The most successful mechanism that has been transferred in this way is that

of trail-laying ants which has been employed, in a number of technical applications [4, 3, 27, 23, 6, 5, 20, 29]

From such case studies, we may understand how specific natural systems work. However, transferring these mechanisms to distributed computing systems proves difficult. Biological systems solve very specific problems and their components are situated and interact in the physical world using air temperature, light signals, chemicals, etc. The difference in abstractions between the biological world and the domain of computing systems is rather large. Therefore, many biological case studies have only limited value for the engineering of concrete software systems.

In this paper, we present the first case study of a self-organizing software system that is not based on biological models. This system, called *Ad hoc Service Grid* (ASG) [14, 15, 16], originates from the domain of distributed computing systems and is situated in the area of Ambient Intelligence. It provides mechanisms for the self-organized replication and placement of services in such an environment. Our emphasis in this study lies on the analysis of the emergent effects perceivable in the ASG. We show how a small set of local rules, executed by service replicas, results in cost-efficient and adaptive global replica placement patterns. It turns out that these simple rules lead to qualitatively different interaction schemes at a higher level of abstraction. Based on the description of the low-level algorithm, we will derive and investigate the mechanisms that lead to these higher-level emergent effects. Furthermore, we generalize the mechanisms that contribute to the emergence in our system, and we propose a catalog of general design questions that can help engineers in dividing the complex problem of creating a self-organizing software system into different aspects.

The rest of this paper is structured as follows: In the next section, we discuss some related work. In Section 3, we introduce the ASG model, including a discussion of the algorithm for self-organized replica placement. Subsequently, we analyze the high-level behavior that can be observed in the overall system. In Section 4, we investigate the relation between the lower-level actions taken by replicas and the emergent higher-level behavior of the overall system. We reveal hidden interactions between replicas and give a detailed discussion of the mechanisms that take effect in the emergence of the higher-level behavior. In Section 5, we generalize these qualitatively new concepts and present the resulting catalog of design questions. We present our conclusions in Section 6.

2 Related Work

The case studies that are at the basis of self-organizing software systems research are conducted by biologists, physicists, and mathematicians. They reveal the mechanisms by which certain natural systems achieve self-organized behavior. This includes, for example, pattern formation in ant colonies [30], swarm flocking behavior [33], thermo regulation in honey bee nests [22], the synchronized flashing of fireflies [31], and the aggregation behavior of slime molds [25], just to name a few. These studies investigate the inherent feedback mechanisms and diverse communication patterns based on light emission, air temperature fluctuation, pheromones, and the like, establishing abstract models of the respective natural systems.

These models inspire computer scientists to apply similar mechanisms, for example, to radio frequency allocation [32], computer security [10], network routing [4], document clustering [28], and optimization problems [5]. Most of these solutions remain rather isolated, representing approaches to very specific problems in computer science.

Attempts to categorize the plethora of nature-inspired mechanisms and to make them accessible as a kind of self-organization tool box have been made by Parunak [21] and, more recently, by Mamei et al. [24]. They try to create a taxonomy that allows an assessment of arbitrary computer science problems such that the most fitting method may be picked to solve them. This is a qualitatively new step as it proposes a more general attempt to engineering self-organizing software systems. Albeit, it is still a bottom-up approach that relies on existing systems as example cases.

Top-down approaches are rare. For example, Gershenson proposes a general methodology for designing and controlling self-organizing systems that introduces mechanisms for "reducing friction" and "promoting synergy" between a system's elements [11]. Such concepts are valuable since they further our general understanding of self-organization. However, it is quite hard to translate them into more tangible ones related to a concrete problem. Wright et al. present a measure of self-organization and propose to use this measure in order to let a genetic algorithm evolve a given system towards self-organization [35]. However, the validity of this modeling approach has only been shown for abstract scenarios. Again, this approach provides additional insight, but it is hard to apply it in practice.

The work presented in this paper is a part of the bottom-up strand. However, the system we investigate originates from the domain of computing systems. Therefore, the concepts do not need to be transformed in any way and are much more easily applicable to other computing systems. Moreover, we do not simply present yet another design pattern. Instead, we attempt to generalize a number of important aspects found in the design of the presented system. These aspects are meant to support a top-down engineering methodology by separating the monolithic view into subviews that may greatly facilitate the engineering process. Thus, our hope is to close the gap between bottom-up and top-down approaches a little bit from both ends.

3 Ad hoc Service Grid

In recent years, the vision of *Ambient Intelligence* (AmI) [7] has produced considerable research efforts. The main idea of AmI is that mobile users may wirelessly access services (anywhere and anytime) that support them in their daily activities without putting any administrative burden on them. One essential building block of AmI are infrastructures that allow local access to local, facility-specific services. To enable services that enhance the user's interaction with his current environment, the Ad hoc Service Grid enriches the physical surrounding of the user (e.g. at shopping malls, construction sites, and trade fares) with dedicated computing resources that enable service provisioning. One example scenario is a shopping mall that offers *ambient services* to customers, enabling them to navigate through the mall, find certain products quickly, and optimize the contents of their shopping cart, for example, according to the overall price or quality. Such services pertain to the local environment of the user and are provided by the local ASG resources that have been distributed over the shopping mall.

The Ad hoc Service Grid infrastructure [16] is easy to setup, flexibly scalable, and requires minimal administrative effort. The easy setup and the flexibility at the networking layer is achieved by adopting *Mobile Ad hoc Networking* (MANET) technology. To minimize the administration overhead, we have created a software layer that we call *ASG Serviceware*. This software has the task of providing basic support for running services within an ASG in a self-organized fashion. That is, all aspects of executing services and adapting to changing user demands shall be managed autonomously by this software.

In the following sections, we briefly describe the relevant aspects of the existing ASG system in preparation of our analysis of its emergent properties. A very detailed description of the ASG including the full experimental setup and many quantitative evaluations can be found in [15].

3.1 System Model

The ASG is based on the concept of *Service Cubes* (also called *Cubes* or *nodes* hereafter). A Service Cube is a PC-class computer dedicated to service provisioning that provides ad hoc networking capabilities and computing power. It has no peripheral devices (display or input devices), relies on a permanent power supply, and is equipped with a wireless network interface. In order to cover a given location with Ambient Services, a number of these Service Cubes is distributed over the location such that they can spontaneously set up a wireless multi-hop network connecting all Cubes. Thanks to the concept of self-contained Service Cubes, an ASG network has a highly modular structure: New modules (Cubes) can be added and existing ones can be removed or repositioned to re-shape the network in an ad hoc fashion. This provides a flexible way of scaling an ASG to an appropriate size and allows for a quick and easy setup. No extensive planning and no construction work is necessary to cover a location in this way. Clients that wish to use ASG services, enter the facility and automatically become a part of the ad hoc network since their mobile device connects to any of the Cubes in an ad hoc fashion. We assume the existence of some technology that allows for a seamless hand-over such that clients may move freely through the ASG network and always stay connected to at least one Cube.

3.2 Self-Organized Replica Placement

An ASG network may take an arbitrary shape and may consist of different numbers of Cubes. Furthermore, the network shape and the number of Cubes as well as the volume of clients, their location, and their demand may vary over time. Temporary network partitions may occur regularly and have profound influence on service availability. In addition to these dynamics, the transmission capacity of the wireless medium is the most scarce resource in such a system. In order to cope with the dynamics and to relieve the network from unnecessary traffic, the ASG Serviceware employs a mechanism for the dynamic replication and placement of services inside an ASG network.

To implement this mechanism, we have developed a purely decentralized service replication and placement algorithm [16, 15]. A service may consist of one or more replicas, and each of these replicas runs the algorithm locally in regular intervals to find a better position within the ASG network. A group of replicas that repeatedly executes the algorithm, distributes itself within an ASG network such that the global communication costs are minimized.

In this case study, we analyze this emergence of global, cost-efficient placement patterns from the local actions of the individual replicas. Despite the fact that there is no global process or knowledge in the system, and even though replicas do not communicate in order to coordinate their placement decisions, global coordination is achieved. The whole system of replicas converges to a stable, low-cost placement that covers the entire network. Moreover, if significant changes occur due to the afore-mentioned dynamics, the distributed algorithm adapts the global placement accordingly to re-establish a cost-efficient placement.

However, as we will see in the following description of the algorithm, the lower-level actions taken by replicas do not directly imply this higher-level behavior of the overall system. How does an adaptive global placement emerge? What are the hidden mechanisms that lead to these patterns?

3.3 Distributed Placement Algorithm

The placement algorithm allows a replica to *migrate* from one node to another, to *replicate* (clone itself and subsequently migrate both clones), or to *dissolve* (remove itself from the system). The immediate goals of every individual replica are to:

1. Maintain a balance between the flows of requests coming in via the set of neighbor nodes. This is achieved using migrations.
2. Avoid that requests have to travel large distances between their sender (client) and the receiving service replica. This is done by creating new replicas.
3. Avoid idle replicas. This is done by dissolving idle replicas.

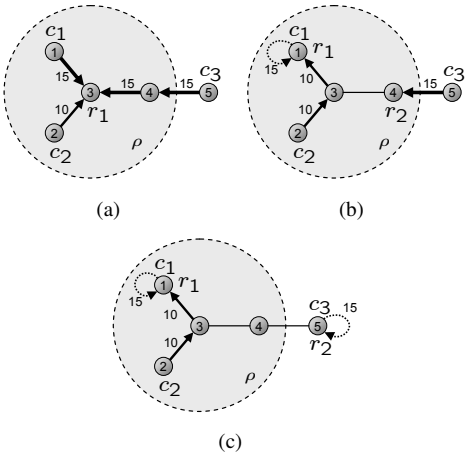


Figure 1. Using the replication radius to reduce the distance spanned by requests.

Let us assume that the ASG network is a graph $G = (N, E)$ and that a replica r_1 is running on node $v \in N$. v has a set of neighbor nodes N_v that contains all nodes in N that are direct neighbors of v. All requests received by r_1 are routed to v via some node in N_v. A series of requests coming in via a neighbor is also called a *message flow*. Such a message flow is called *dominant* iff its volume (number of messages per time unit) is greater than the accumulated volume of all remaining message flows entering v. A flow is called *significant* if there is no single incoming flow with a greater volume. We also call the respective neighbor nodes *dominant* or *significant* respectively. Each instance of the algorithm constantly inspects the volumes of the incoming message flows of its replica and periodically takes a local adaptation decision based on this observation. The algorithm has three rules for the possible adaptations:

1. **Idle Rule:** A replica dissolves (is removed) if it receives less then α requests in a time interval m.
2. **Replication Rule:** A replica replicates to neighbor nodes u and w, if it receives a significant message flow via u that consists of requests with an average path length of more than ρ hops. An example is depicted in Figure 1(a) where v is node 3, u is node 4, and $\rho = 1$. w is either set to the dominant node among the remaining neighbors (node 1 in Figure 1(a)), or the second replica stays at its current node if such a dominant node does not exist. ρ is called the *replication radius*.
3. **Migration Rule:** A replica migrates to a neighbor node u, if the message flow that is coming in via u is dominant. Figure 2(a) shows an example where r_1 is running on node 3, and node 4 is the dominant node.

The placement algorithm invokes these rules in the order in which they are listed above. In each run, the first rule whose condition triggers is executed, and the remaining ones are skipped. When a new service is installed on an ASG, it starts with a single, initial replica that is put on an arbitrary node.

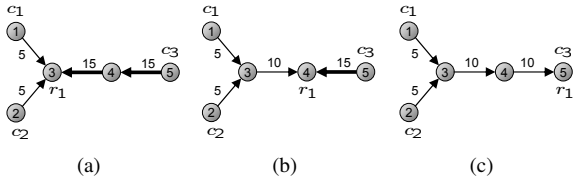


Figure 2. Exploiting dominating flows for incremental cost optimization.

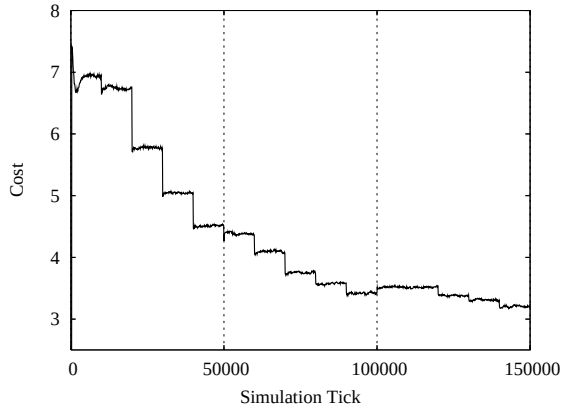
Of course, the success of the above adaptations is heavily depending on the clients' choice of replicas. Our basic assumption is that clients always choose the replica that is closest to them in terms of network hops. This is enforced by the distributed ASG lookup service [17]. The instances of this service are distributed in the ASG network such that each client has at least one instance in its one-hop neighborhood. A query to this instance yields the service replica that is closest to the client's current position.

3.4 Higher-Level Behavior

Our main goal is to use as little wireless bandwidth as possible since this is regarded as being the most critical resource in the ASG. Therefore, we evaluate the ASG system based on the communication costs. These costs are defined as the *number of message transmissions per time unit* that is necessary to serve all client requests. If a client sends a

request to a service that is n network hops away, this results in n transmissions of the request message.

The first global observation that can be made when the system is running is that the overall communication costs are successively reduced. Figure 3 shows how the overall costs produced in the system per simulation tick decreases over time to less than 50% of the initial value. The algorithm is run every 10.000 ticks, and with each run, there is a notable drop in costs. Additionally, every 50.000 ticks, the client request patterns are changed. This is done by recomputing the probabilities with which a client that is connected to a node w sends a request for the service. Due to the continuous attempt to maintain a good placement, the initial increase in cost that is caused by such perturbations is countered, and the cost reduction continues.



(a) Initial phase

Figure 3. Communication cost optimization.

Figure 4 depicts an example of an ASG network that runs a service (nodes that host replicas are black). The figure shows how the replicas spread out to cover the network. Changes in client request patterns are indicated by marks in the upper left corner of the snapshots. Note how the replicas adapt quickly (within 3 iterations of the placement algorithm) to changes and retain a stable placement until the next perturbation occurs. After the third perturbation, stability is maintained until perturbation number eight is applied to the system.

4 Emergence of a Global Replica Placement

The quick reduction of the costs and the adaptiveness displayed by the higher-level behavior of our system are, of course, the result of the actions collectively taken by the replicas. However, the transition from the replicas' actions at the lower level to the global behavior at the higher level (Figure 5) is not obvious at first glance. How are replicas connected to each other? How do they interact to achieve

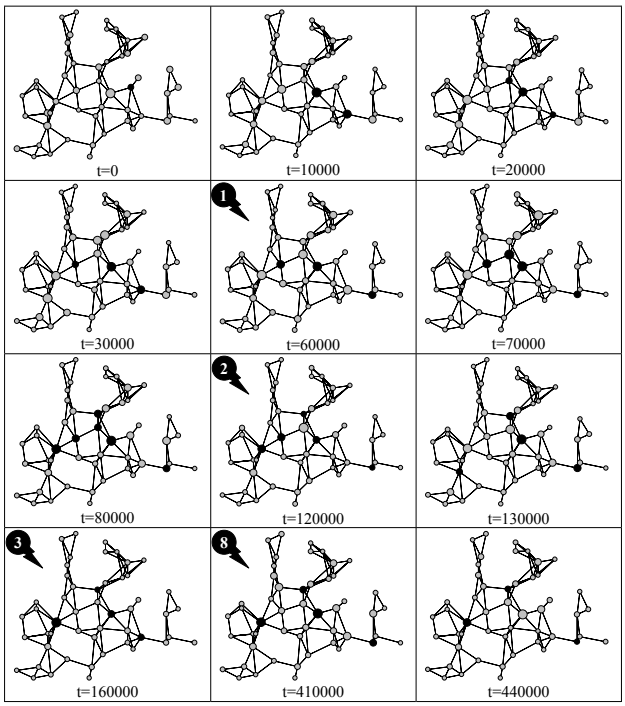


Figure 4. Placement snapshots.

global coordination? After all, the actions taken by replicas appear to be rather self-centered and isolated from each other. Why do migrations towards dominant neighbors and replications towards long request paths actually generate a stable, cost-efficient, adaptive, global replica placement?

To fully understand how this transition occurs, we have to consider concepts that are rather different from those used to describe the placement algorithm and that fill the gap between the two levels depicted in Figure 5. In this section, we will investigate what happens *behind the scenes* through the interactions taking place in the system. We will show that feedback plays a vital role in the emergence of the global behavior and how a certain form of competition among replicas controls their global distribution.

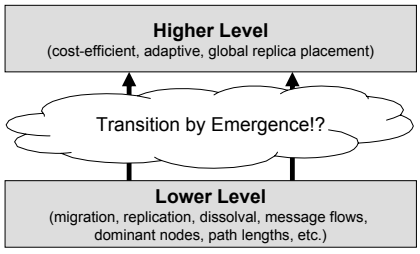
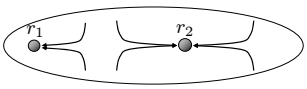


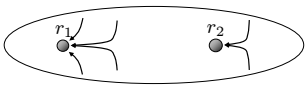
Figure 5. Emergence: The big picture.

4.1 Migration – Basic Feedback Process

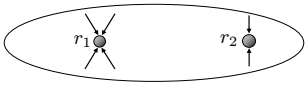
It is intuitively clear that the algorithm's migration rule pushes a replica towards the bulk of clients that send requests to it. In combination with the fact that the ASG lookup service always tries to assign the closest replica to any client, this creates a feedback loop. This whole feedback process has three phases that are depicted in Figure 6: Let us assume that a given ASG network (denoted by the oval) is served by two replicas r_1 and r_2. Initially, r_1 is in a state of imbalance since it receives client requests (indicated by the arrows) from a single direction (Figure 6(a)). The migration rule will force r_1 to migrate towards this direction. By this movement, r_1 suddenly becomes the closest replica for a set of clients that sent their requests to r_2 before (Figure 6(b)). Thus, by moving towards its clients, r_1 attracts more clients which, in turn, pulls r_1 even more towards the direction of client requests. This is a positive feedback effect which takes place in the attraction phase and drives the system towards a new configuration quickly. After a while, r_1 enters the region where its clients are positioned and the requests start coming in via more than one neighbor node. This decreases the attraction until the dominant message flow depletes and the motion of r_1 stops. This *depletion phase* (Figure 6(b)) introduces a negative feedback process that has a strong stabilizing effect as it slows down r_1 quickly. In Figure 6(c), the system has reached an equilibrium state where no distinct *force* is applied to either of the replicas. Moreover, the average length of the request paths, and therefore the costs, are minimal when our example system is in this state. In the *fluctuation phase*, the system stays in this state until notable changes or fluctuations occur in the request patterns of clients that lead to an imbalance again and restart the adaptation process.



(a) Attraction phase



(b) Depletion phase



(c) Fluctuation phase

Figure 6. The three feedback phases.

Figure 6 clearly shows how replicas are coupled and interact indirectly via message flows. Each message flow is consumed by some replica, and each time a flow is redirected to a different replica (due to changes in the distance to clients), another replica potentially enters a state of imbalance since it looses this very message flow. In our example, r_2 is thrown out of its equilibrium state as r_1 attracts a portion of its clients 6(a). Therefore, r_2 is also forced to migrate in order to regain its equilibrium (Figure 6(c)).

In general, the migration rule applies a force to the system that leads to an automatic adaptation each time a deviation from the equilibrium state occurs.

4.2 Replication and Removal – Adapting to the Network

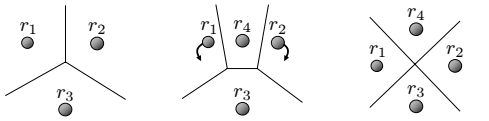
The replication rule applies a different kind of force to the system of replicas: The replication radius ρ leads to the creation of replicas until each replica covers a portion of the network with a radius of about ρ. Therefore, this rule leads to an adaptation of the number of running replicas to the size of the network. If the network grows and clients start sending requests from the new regions, the system automatically creates new replicas to cover them. This ensures that client requests travel at most ρ hops on average, which, in turn, controls the overall communication costs.

The idle rule is the counterpart of the replication rule in the overall mechanism for the control of the number of running replicas. If the number of clients diminishes, then less requests are issued throughout the network. Therefore, less replicas have to be maintained in order to serve these requests. Some replicas, for which the number of requests falls below the threshold of α, will then be automatically removed, and their clients start using other replicas. Due to the indirect coupling between the replicas, this may lead to migrations among the remaining replicas in order to compensate the possible imbalance caused by this change. The idle rule may also be viewed as a sort of *garbage collection* as it constantly removes unused replicas.

4.3 Adaptive Cell Structuring

Due to the the fact that clients always use the closest replica, the collective adaptation process of all replicas creates a global pattern consisting of the cells of a Voronoi decomposition. In each cell, a replica serves all requests being issued inside the cell. The replica placement algorithm replicates and migrates the set of replicas such that a stable configuration of Voronoi cells is created.

Figure 7 depicts how the cell structure changes if the replica configuration is changed. For this simplified example, we assume that the plane is densely populated with nodes and that client requests are issued uniformly from all nodes. Figure 7(a) shows a stable Voronoi cell pattern with three replicas: Each replica is placed at about the center of its cell and, thus, the forces applied by client requests are



(a) Stable (b) Disturbance (c) Re-stabilized

Figure 7. Adaptation of the cell structure.

about the same in all directions. Let us assume that, due to some disturbance in the request patterns, a new replica r_4 is created between r_1 and r_2. The resulting Voronoi diagram is shown in Figure 7(b). This disturbance changes the cells of r_1 and r_2 such that they are close to the borders of their cells. Since the requests coming from the upper part of the plane are now redirected to r_4, they start receiving an unbalanced flow of requests. According to the migration rule, r_1 and r_2 are forced to move to a position where the flows are in balance again. The resulting four-cell setup is depicted in Figure 7(c). Thus, a reconfiguration on a global scale is caused by a local disturbance without any direct communication taking place between the replicas.

4.4 Adaptation Cycle and Attractors

As we have seen in the qualitative analysis of the hidden effects of our simple replica placement algorithm, it introduces an *adaptation cycle*. It implicitly defines the notion of a stable state as follows:

Definition 1 (Stability Condition) *The state of the overall replica system is stable iff*

1. *the incoming message flows of all replicas are balanced (i.e. there is no dominant node),*
2. *no replica serves requests that average a path length of more than ρ hops, and*
3. *each replica receives a sufficient amount of requests per time unit.*

In such a state, non of the adaptation rules is triggered and, thus, the current replica placement is retained.

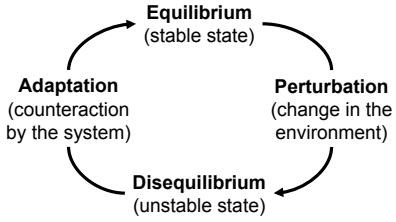


Figure 8. Adaptation cycle.

Perturbations due to changes in client behavior or network topology can push the system into an unstable state in which the stability condition is violated. This automatically leads to adaptations through the respective rules of the distributed placement algorithm until a new stable state is reached (cf. Figure 8). This resembles a classical *control loop* that counters every deviation from the prescribed *set point*. However, in our case, the set point is no particular placement or any geometrical characteristics thereof, as one may expect. Instead, the set point is *stability*. In complex systems terminology, this set point is also called an *attractor* [26]: Let α be a replica placement and let β a state of the environment that the replica system has to adapt to (client request pattern, network topology etc.). Among the set S of all possible system states (α, β), there is a subset $S_{stab} \subseteq S$ that satisfies the stability conditions:

$$S_{stab} = \{(\alpha, \beta) \in S \mid \alpha \text{ fulfills Def. 1 under } \beta\}$$

Through the adaptation cycle, the replica system is constantly attracted to some placement α^i such that $(\alpha^i, \beta^j) \in S_{stab}$ for the current environment state β^j. If, through some sort of dynamic change, the environment enters a different state β^k such that $(\alpha^i, \beta^k) \notin S_{stab}$, then this instability automatically triggers an adaptation and pushes the replica system into a new state α^l such that $(\alpha^l, \beta^k) \in S_{stab}$ holds again until the next environmental change.

But how is stability related to our original goal of reducing the communication cost in the overall system? For each individual replica, a stable state implies that the local costs cannot be immediately reduced by an adaptation. As indicated in Figure 2, a migration towards a dominant node has a potential for reducing the local costs since the sum of request flows received from the remaining nodes ($\mathcal{F}_{rest}$) may have to travel one hop further while the hop count of the dominant flow ($\mathcal{F}_{dom}$) is reduced by one. Due to the definition of dominance, we have $\mathcal{F}_{rest} < \mathcal{F}_{dom}$, and the number of transmissions per time unit required to transport the client requests to the migrated replica is reduced by $\mathcal{F}_{dom} - \mathcal{F}_{rest} > 0$. In a stable state, a replica does not have this immediate possibility of reducing the local costs. Thus, the costs are the lowest that are currently achievable by the replica. There may well be possible migrations that eventually lead to a reduction. However, the replica may only speculate about them. Therefore, the stable state represents a local cost minimum for this particular replica. The communication cost in the overall system is the sum of all local costs. Thus, the overall costs are at a local minimum if all replicas are in a stable state. In other words, the system is constantly forced into some cost-efficient attractor. This attractor state may not represent the overall global cost minimum, but this is the best we can achieve with a purely distributed heuristic approach in a dynamic environment.

5 Generalization of the Results

In the preceding qualitative analysis of our simple replica placement system, we have shown that its complex high-level behavior is a consequence of the interactions among the individual replicas. Replicas are coupled and interact indirectly through the message flows that are an attribute of their environment (network topology and client behavior). This interaction is much more complex than is indicated by the rather simple placement algorithm. It can be explained using concepts like feedback, adaptive cells, adaptation cycles, and attractors that fill the *semantic gap* (depicted in Figure 5) between the lower-level actions and the higher-level behavior.

This case study has its own value in the same sense as studies of biological systems do: It may inspire new technological solutions that solve related problems with similar self-organization approaches. However, the concepts and technologies employed here are much closer to those of systems that may potentially exploit them in different information technology contexts.

Beyond being a blueprint, our system can be valuable with respect to another important question: *Can this study give some indications as to how such a self-organizing system may be purposefully engineered in general?*

We cannot derive a complete engineering process. However, the self-organized replica placement system has some characteristics and features that may be applicable in a more general context. In the following, we analyze these general features and pose some questions that should be answered by engineers in order to create similar characteristics in other systems. The aim is to divide the immensely complex task of creating self-organization into a number of interrelated subtasks that a software engineer may tackle individually. This reduces the required effort and may lead to a faster decision about the applicable mechanisms, or it may also lead to the early recognition of the fact that a given problem does not lend itself for a self-organizing solution at all. Moreover, the concepts that should be applied in the design and the vocabulary that is necessary to argue about them differ dramatically from those used in conventional software engineering. The following *catalog of design questions* and the concrete example of the ASG system should help in adjusting to this new language.

- **Self-organization and adaptation:** The self-organization process described in this paper has the inherent goal of adapting the system to changing external conditions in order to preserve its cost-efficiency. There may be other self-organization processes that do not necessarily aim at adaptation. However, we think that in the area of self-organizing software systems, adaptation is always an inherent requirement.
  - *What is the subject of adaptation in the system?*
- **Adaptation and optimization:** Adaptation may be defined as a continuous optimization process. An adaptive system recognizes any shift towards suboptimal states (usually caused by a changing environment) and counters this shift by optimizing the system's state to fit the new environment. This form of optimization is an online task. It has to be repeated constantly, and it has to lead to some solution in a timely fashion in order for the system to stay operational. In this sense, it is beneficial to think of the nature of this adaptation process at the beginning of any design phase.
  - *What is the optimization criterion?*
  - *Which mechanisms can be applied in order to adapt (optimize) the system?*
  - *Is the optimization task very complex?*
- **Optimization vs. acceptability:** The nature of the systems that we regard is such that an optimal solution will be achievable only in rare cases. This is due to the fact that we deal with decentralized online optimization. Realistically, the best we can hope for is an *acceptable* solution. This may represent a problem for some systems whose survival depends on their optimal configuration.
  - *Is an acceptable solution good enough for a given system?*
- **Local and global optimization:** One of the beneficial features that is commonly assigned to self-organizing systems is their decentralization [19, 18]: The actors in a self-organizing system act locally, and their view is restricted to their immediate neighborhood. In terms of optimization, this imposes a constraint upon the system: A acceptable (low-cost) global configuration must be achievable through a combination of locally acceptable solutions. Or, more generally, a mapping of the local to the global acceptability criterion must exist. In our system, every replica tries to reach an equilibrium state with respect to the incoming message flows. As we have shown in Section 4.4, the combination of local equilibriums maps to a low-cost global state.
  - *Is there a mapping of the local to the global acceptability criterion in the given system?*
- **Coupling (interaction) via the subject of work:** A key to the global coordination of any decentralized adaptation process can be a local coupling of the actors via the very subject of the adaptation. In the ASG, there is no global coordination involved. Replicas indirectly coordinate their respective locations with their

local neighbors (closest fellow replicas) by influencing each other's message flows (cf. Sections 4.1 and 4.3). At the same time, these message flows are subject to the optimization task. This general principle is called *Stigmergy* [13]. In the same way as termites coordinate their nest-building actions via the structures they build, replicas coordinate their placement activities via the message flows that they try to balance. Discovering and exploiting such a coupling can greatly simplify the design of a concrete self-organization mechanism. It should be noted, that some form of indirect interaction exists in most cases, and it is important to understand its character in any case.

  - *Is stigmergic coordination possible in the given system?*
  - *What is the subject of optimization, and are the actors coupled by it?*

- **Feedback process:** The simple feedback process in the system evolves from *(i)* replicas moving towards the clients that send requests and *(ii)* the clients always choosing the closest replica. This process involves positive feedback that leads to a fast convergence to a cost-efficient state as well as negative feedback that stabilizes the system when such a state is approached. This mechanism is the driving force behind the concept of self-organized replica placement. Similar processes are common in natural self-organizing systems. Thus, it can be beneficial to look at a problem that requires self-organization from the feedback perspective.

  - *Is there some feedback mechanism inherent to the problem, or*
  - *can one be designed for the given system?*

- **Stable attractors:** The system must have attractor states that are approached by virtue of the self-organization algorithm. Moreover, these attractors must be cost-efficient with respect to the cost function of the system, and they must be stable in the sense that the attractiveness of such a state must not disappear as a consequence of the system entering it. This would result in a system that constantly approaches good states without ever reaching one. Metaphorically speaking, the resulting system should behave like a ball that is constantly rolling down-slope towards low-cost states by itself with respect to the acceptability criterion. The answers to the following questions are not obvious in most cases. Moreover, they heavily depend on the definition of the system as well as on the criterion for optimality or acceptability.

  - *Does the given system have stable attractors?*
  - *How can these be approached locally through the adaptations of the actors within the system?*

6 Conclusions

The current approach to engineering self-organizing software systems is heavily relying on copying mechanisms found in biological systems. While this can be fruitful, we argue that computer science should start developing its own core concepts for this task. In this paper, we have presented the first case study of an original self-organizing distributed software system that employs concepts originating from the distributed system domain.

We have investigated the mechanisms and concepts that lead to the emergence of a globally coordinated replica placement from simple and local actions executed by individual service replicas. These concepts fill the gap that exists between the lower-level actions of the components in the system and the higher-level complex behavior of the overall system. Like any existing case study, this one can be used as a blueprint for the design of similar concepts in comparable systems. Subsequently, we have generalized the concepts leading to the emergence in the ASG system in an attempt to present more general guidelines for engineering self-organizing software systems. Our catalog of design questions helps in structuring the design process and supports the engineer in dealing with the central problems involved with achieving self-organization. It serves as a conceptual framework rather than a formal engineering process. However, it divides the complex task of engineering a self-organizing software systems into subtasks that are much more easily handleable. Moreover, it helps in deciding if a given problem can be solved by a self-organizing software system.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, 2002.

[2] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[3] L. Carrillo, J. L. Marzo, P. Vilà, and C. A. Mantilla. MAntS-Hoc: A Multi-Agent Ant-Based System for Routing in Mobile Ad Hoc Networks. In *Proceedings of Setè Congrés Català d'Intelligència Artificial (CCIA'2004)*, pages 285–292, 2004.

[4] G. Di Caro and M. Dorigo. Ant Colonies for Adaptive Routing in Packet-Switched Communications Networks. In Eiben et al. [8], pages 673–682.

[5] M. Dorigo, G. D. Caro, and L. M. Gambardella. The Ant Colony Optimization Meta-Heuristic. *Artificial Life*, 5(2):137–172, 1999.

[6] M. Dorigo, G. D. Caro, and M. Sampels, editors. *Ant Algorithms, Third International Workshop, ANTS 2002, Brussels, Belgium, September 12-14, 2002. Proceedings*, volume 2463 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[7] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for Ambient Intelligence in 2010. Technical Report, The IST Advisory Group (ISTAG), 2001.

[8] A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature – PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, volume 1498 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[10] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer Immunology. *Communications of the ACM*, 40(10):88–96, Oct. 1997.

[11] C. Gershenson. A General Methodology for Designing Self-Organizing Systems. Technical Report 2005-05, ECCO, May 2005.

[12] S. Giordano. *Mobile Ad hoc Networks*, pages 325–346. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[13] P. P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes sp.* La théorie de la stigmergie: Essai d'interpretation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959.

[14] K. Herrmann. Ad hoc Service Grid – Self-Organizing Distribution of Ambient Services. Report PTB-IT-13, Physikalische Technische Bundesanstalt, Wirtschaftsverlag NW, Verlag für neue Wissenschaft GmbH, Mar. 2006. (ISBN 3-86509-509-7).

[15] K. Herrmann. *Self-Organizing Infrastructures for Ambient Services*. Phd thesis, Berlin University of Technology, July 2006.

[16] K. Herrmann, K. Geihs, and G. Mühl. Ad hoc Service Grid – A Self-Organizing Infrastructure for Mobile Commerce. In *Proceedings of the IFIP TC8 Working Conference on Mobile Information Systems (MOBIS 2004)*. IFIP – International Federation for Information Processing, Springer-Verlag, Sept. 2004.

[17] K. Herrmann, G. Mühl, and M. A. Jaeger. A Self-Organizing Lookup Service for Dynamic Ambient Services. In *25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pages 707–716, Piscataway, NJ, USA, June 2005. IEEE Computer Society Press.

[18] K. Herrmann, M. Werner, and G. Mühl. A Methodology for Classifying Self-Organizing Software Systems. *International Transactions on Systems Science and Applications*, 2(1):41–50, Sept. 2006.

[19] F. Heylighen and C. Gershenson. The Meaning of Self-organization in Computing. *IEEE Intelligent Systems*, 18(4):72–75, July – Aug. 2003.

[20] K. M. Hoe, W. K. Lai, and T. S. Tai. Homogeneous Ants for Web Document Similarity Modeling and Categorization. In *Proceedings of the Third International Workshop on Ant Algorithms (ANTS 2002)*, Sept. 2002.

[21] H.V.D.Parunak. Go to the Ant: Engineering Principles from Natural Multi-Agent Systems. *Annals of Operations Research*, 75:69–101, 1997.

[22] J. Jones, M. Myerscough, S. Graham, and B. Oldroyd. Honey Bee Nest Thermoregulation: Diversity Promotes Stability. *Science*, 305:402–404, 2004.

[23] I. Kassabalidis, M.A.El-Sharkawi, R. II, P. Arabshahi, and A.A.Gray. Swarm Intelligence for Routing in Communication Networks. In *Proceedings of the IEEE Globecom 2001*, Nov. 2001.

[24] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case Studies for Self-organization in Computer Science. *Journal of Systems Architecture*, 2006.

[25] A. F. M. Marée and P. Hogeweg. Modelling Dictyostelium Discoideum Morphogenesis: The Culmination. *Bulletin of Mathematical Biology*, 64(2):327–353, Mar. 2002.

[26] J. Milnor. On the Concept of Attractor. *Communications in Mathematical Physics*, 99(2):177–195, 1985.

[27] Y. Ohtaki, N. Wakamiya, M. Murata, and M. Imase. Scalable Ant-based Routing Algorithm for Ad-hoc Networks. In *Proceedings of the 3rd IASTED International Conference on Communications, Internet, and Information Technology (CIIT 2004)*, 2004.

[28] V. Ramos and J. J. Merelo. Self-Organized Stigmergic Document Maps: Environment as a Mechanism for Context Learning. In *AEB'2002 – 1st Spanish Conference on Evolutionary and Bio-Inspired Algorithms*, pages 284–293, Feb. 2002.

[29] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like Agents for Load Balancing in Telecommunications networks. In *Agents '97, Proceedings of the First International Conference on Autonomous Agents*, pages 209–216, New York, NY, USA, 1997. ACM Press.

[30] R. V. Sole, E. Bonabeau, J. Delgado, F. P., and J. Marin. Pattern Formation and Optimization in Army Ant Raids. *Artificial Life*, 6(3):219–226, 2000.

[31] S. H. Strogatz and I. Stewart. Coupled Oscillators and Biological Synchronization. *Scientific American*, 269(6):102–110, Dec. 1993.

[32] R. Tateson. Self-organising Pattern Formation: Fruit Flies and Cell Phones. In Eiben et al. [8].

[33] J. Toner and Y. Tu. Flocks, Herds, and Schools: A Quantitative Theory of Flocking. *Physical Review E*, 58(4), 1998.

[34] A. Vakali and G. Pallis. Content Delivery Networks: Status and Trends. *IEEE Internet Computing*, 7(6):68–74, Nov. 2003.

[35] W. A. Wright, R. E. Smith, M. Danek, and P. Greenway. A Generalisable Measure of Self-Organisation and Emergence. In *Proceedings of the International Conference on Artificial Neural Networks – ICANN 2001*, volume 2130 of *Lecture Notes in Computer Science*, pages 857–864. Springer-Verlag, Aug. 2001.