Towards a Theory of Self-Organization

Emmanuelle Anceaume[1] Xavier Défago[2] Maria Gradinariu[1] Matthieu Roy[3]

[1] IRISA, Campus de Beaulieu, 35042 Rennes CEDEX, France
[2] JAIST and PRESTO, JST, Japan
[3] LAAS-CNRS, France

Abstract. This paper aims at providing a rigorous definition of *self-organization*, one of the most desired properties for dynamic systems, such as peer-to-peer systems, sensor networks, cooperative robotics, or ad-hoc networks. We propose a framework in order to prove the self-organization of dynamic systems with respect to generic criteria (e.g., similarity, load balancing, geographical neighborhood, battery level) that can be composed in order to construct more complex criteria. We illustrate our theory with a case study that consists in proving the self-organization of CAN, a representative peer-to-peer system.

1 Introduction

Self-organization is an evolutionary process in which the effects of the environment are present. Natural phenomena, living forms, or social systems (e.g., growing crystals, cells aggregation, ant colonies) are examples of self-organizing systems in which a global order of the system emerges from local interactions.

In the newly emerging fields of distributed systems (p2p, ad-hoc networks, sensor networks, cooperative robotics), self-organization became one of the most desired properties.

The major feature of all recent scalable distributed systems is their extreme dynamism in terms of structure, content, and load. In p2p networks, nodes continuously join and leave the system. In large scale sensor, ad-hoc or robot networks, the energy fluctuation of batteries and the inherent mobility of nodes induce a dynamic aspect of the system (the system size and the topology may change) that must be addressed. In all these systems there is no central entity in charge of their organization and control, and there is an equal capability, and responsibility entrusted to each of them to own data [10]. To cope with such characteristics, these systems must be able to spontaneously organize toward exhibiting desirable global properties. In peer-to-peer systems, self-organization is handled through protocols for node arrival and departure, based either on a fault-tolerant overlay network, such as in CAN, Chord, Pastry [7, 14, 18, 16], or on some localization and routing infrastructure, such as in OceanStore [11, 22]. Recent peer-to-peer applications exploit the natural self-organization of peers in semantic communities (clusters) [9, 12, 17]. In ad-hoc networks, solutions have been proposed for a self-organizing public-key management system that allows users to create, store, distribute, and revoke their public keys without the help of any trusted authority or fixed server [4]. Self-organization was also used in order to cluster ad-hoc nodes [21]. Self-organizing algorithms have also been developed that arrange mobile robots into predefined geometric patterns (e.g., [19]). Inspired from crystal growth, Fujibayashi et al. [8] simulations self-organizing heuristics for the shape formation of a group of mobile robots. Their work uses the notion of virtual spring—a virtual link between neighboring nodes. The global shape is obtained by tuning the parameters of the springs.

Informal definitions for self-organization, or the related self* properties (e.g., self-configuration, self-healing or self-reconfiguration) have been proposed previously [3, 20, 21]. Babaoğlu et al. [3] propose a platform, called Anthill, aimed at the design of peer-to-peer applications based on self-organized colonies and swarms of agents. Anthill offers a bottom-up opportunity to understand the emergent behavior of complex adaptive systems. Walter et al. [20] focus on the concept of reconfiguration of a metamorphic robotic system with respect to a goal configuration. One of the problems left open in this work is the specification of admissible and non-admissible configurations, key notions in proving the correctness of the proposed solutions. Zhang and Arora [21] propose the concepts of self-healing and self-configuration in wireless ad-hoc networks, and propose self-stabilizing [5] solutions for self* clustering in ad-hoc networks.

The correctness proofs for all previously mentioned self-organizing systems should be based on a well-founded theoretical model, able to encapsulate the dynamic behavior of these systems. Dynamic systems must cope with frequent changes in topology and size. Hence, the characterization of the self-organizing aspects of these systems cannot solely focus on the non-dynamic periods, since they may be absent or very short. Moreover, defining self-organization as a simple convergence process towards a stable predefined set of admissible configurations is inadequate for two reasons. First, it may be impossible to clearly characterize the set of admissible configurations since, in dynamic systems, a configuration should include the state of some key parameters that have a strong influence on the dynamicity of the system. These parameters can seldom be quantified *a priori* (e.g., the status of batteries in sensor networks, or the data stored within p2p systems). Second, due to the dynamic behavior of nodes, it may happen that no execution of the system converges to one of the predefined admissible configurations.

The main contribution of this paper is to propose a formal specification of the self-organization notion which, to the best of our knowledge, has never been formalized in the area of scalable and dynamic systems, in spite of an overwhelming use of the term. Our specification is based on the principles that govern dynamic systems. The first one relates to the exchange of information or resources within the system (components are possibly capable to infinitely often retrieve new information/resources from components around them). The second one is the dynamics of these systems (components have the ability to move around, to leave or to join these systems based on local knowledge). The third principle is the specificity of the components: Among all components of the system, some have huge computation resources, some have large memory space, some are highly dynamic, some have broad centers of interest. In contrast, seeing such systems as a simple mass of components completely obviates the differences that may exist between individual components; those very differences that make the richness of these systems.

The tenets mentioned above share as a common seed the locality principle, i.e., the fact that interactions and knowledge are both limited in range. We formalize this idea, leading first to the notion of *local self-organization*. Intuitively, a locally self-organizing system should reduce locally the entropy of the system. For example, a locally self-organized p2p system forces components to be adjacent to components that improve, or at least maintain, some property or evaluation criterion. We then formalize the notion of *self-organization* by imposing the system to be locally self-organizing at all its nodes and by ensuring that despite its dynamicity, the system entropy progressively reduces.

The second contribution of this work is a case study. Using our framework we prove the weak self-organization of Pastry CAN, a well known peer-to-peer overlay.

The remaining of this paper is organized as follows: Section 2 proposes a model for dynamic and scalable systems. Section 3 formalizes the local and global self-organization properties. In Section 4, we propose the study of CAN, a dynamic peer-to-peer overlay. Section 5 concludes and discusses open issues.

2 Model

2.1 Dynamic System Model

Communication Graph. The physical network is described by a weakly connected graph. Its nodes represent processes of the system and its edges represent established communication links between processes. The graph is referred in the following as the communication graph. We assume that the communication graph is subject to frequent and unpredictable changes: processes can leave or join the system arbitrarily often, and they can fail temporarily (transient faults) or permanently (crash failures). Communication links can commit transient failures (e.g., messages loss).
Data Model. Nearly all modern applications in the dynamic distributed systems are based on *the principle of data independence*—the separation of data from the programs that use the data. This concept was first developed in the context of database management systems.

In dynamic systems, in particular in P2P systems, data stored locally at each node, organized in flat or hierarchical structures (e.g., XML trees), play a crucial role in creating semantic based communities (logical links between processes that store or query similar data).

Note that system data is subject to frequent and unpredictable changes adjusting to nodes connections and disconnections. Data also suffers modifications like replication, aggregation, removal and can be subject to permanent or transient failures.

Logical Overlay. We consider the network plus the data stored in the network represented by a logical multi-layer overlay, each logical layer l being a weakly connected graph, also referred to as the logical communication graph at layer l. In order to connect to a particular layer l, a process executes an underlying connection protocol. A process p is called *active* at a layer l if there exists at least one process q which is connected at l and aware of p. The set of logical *neighbors* of a process p at a layer l is the set of processes q such that the logical link (p, q) is up (p and q are aware of each other) and is denoted $\mathcal{N}^l(p)$. Notice that a process p may belong to several layers simultaneously. Thus, p may have different sets of neighbors at different logical layers. Can, Pastry or Chord ([14, 15, 18]) are logical overlays using DHTs as design principle. In sensors or ad-hoc networks, connected coverings (such as trees, weakly connected maximal independent sets or connected dominating sets) can also be seen as logical overlays.

2.2 State Machine-based Framework

To rigorously analyze the execution of the dynamic systems, we use the dynamic I/O automata introduced by Attie and Lynch [2]. This model allows the modeling of individual components, their interactions and their changes. The external actions of a dynamic I/O automata are classified in three categories, namely the actions that modify data (by replication, aggregation, removal, or writing), the input-output actions (I/O actions), and dynamic actions (C/D actions for Connection-Disconnection actions) describing the mobility within the system. A configuration is the system state at time t altogether with the communication graph and data stored in the system.

3 Self-Organization

In this section we propose to formally define the notion of self-organization in the context of scalable and dynamic systems (in particular p2p systems) altogether with tools for proving their self-organization. Self-organization strongly relies on the local self-organization property (see Section 3.1), and is characterized by liveness and safety properties (see Section 3.2).

3.1 Local Self-Organization

Intuitively, a locally self-organizing system should force processes to improve or at least maintain some criterion. In the following we restrict our attention to *insensitive criterion*, that is criterion whose evaluation at a process is not modified by the internal actions of other processes. A typical example of such criterion is the proximity metric in the nodeId space. Let $\mathcal{C}$ be a $[0, 1]$-valued function defined on the local neighborhood of a process (the local neighborhood of a process p includes both the state of p and the state of p's neighbors). In the following $\mathcal{C}_p$ denotes an evaluation criterion in the local neighborhood of a process. Let γ_p be a $[0, 1]$ function defined for a process p, a configuration c

and an evaluation criterion $\mathcal{C}_p$. $\gamma_p(c, \mathcal{C}_p)$ is the aggregate of the $\mathcal{C}_p(q)$ values in the configuration c for all one hop neighbors q of p. In the following $\gamma_p(c, \mathcal{C}_p)$ is referred as the local aggregate criterion.

In order to define local self-organization, we introduce the notion of *stable configurations*. Informally, a configuration c is *p-stable* for a given evaluation criterion in the neighborhood of a process p if the local aggregate criterion reached a local maximum in c.

Definition 1 (p-stable configuration). *Let c be a configuration of a system $\mathcal{S}$ and p be a process, $\mathcal{C}_p$ be an evaluation criterion and $\gamma_p(c, \mathcal{C}_p)$ the local aggregate of $\mathcal{C}_p$ at the configuration c. Configuration c is p-stable for γ_p if, for any configuration c' reached from c after one action executed by p, $\gamma_p(c, \mathcal{C}_p) \geq \gamma_p(c', \mathcal{C}_p)$*

Definition 2 (local self-organization). *Let $\mathcal{S}$ be a system, p a process, $\mathcal{C}_p$ an evaluation criterion of p and γ_p the aggregate of $\mathcal{C}_p$. $\mathcal{S}$ is locally self-organizing for γ_p if $\mathcal{S}$ eventually reaches a p-stable configuration. $\mathcal{S}$ is locally self-organizing if $\forall p \in \mathcal{S}$, $\mathcal{S}$ is locally self-organizing for γ_p.*

In p2p systems local self-organization should force processes to be logical neighbors with processes which improve the evaluation criterion. Module 1 executed by a process p, referred in the following LSA, proposes a local self-organizing generic algorithm for an arbitrary insensitive criterion $\mathcal{C}$. Note that existing DHT-based peer-to-peer systems (see Section 4) execute similar algorithms to ensure self-organization with respect to specific criteria (e.g., geographical proximity). The nice property of our generic algorithm is its adaptability to unstructured networks.

LSA is based on a greedy technique, which reveals to be a well adapted technique for function optimization. Its principle follows the here above intuition: Let q such that $q \in \mathcal{N}^{\mathcal{C}}(p)$, and r such that $r \in \mathcal{N}^{\mathcal{C}}(q)$ but $r \notin \mathcal{N}^{\mathcal{C}}(p)$, where $\mathcal{N}^{\mathcal{C}}(p)$ and $\mathcal{N}^{\mathcal{C}}(q)$ are the logical neighborhoods of p and q respectively with respect to the criterion $\mathcal{C}$. If p notices that r improves the evaluation criterion previously computed for q, then p replaces q by r in $\mathcal{N}^{\mathcal{C}}(p)$. Inputs of this algorithm are the evaluation criterion $\mathcal{C}$ and the set of p's neighbors for $\mathcal{C}$, that is $\mathcal{N}^{\mathcal{C}}(p)$. The output is the updated view of $\mathcal{N}^{\mathcal{C}}(p)$. Given a criterion $\mathcal{C}$, a p-stable configuration, in this context, is a configuration where for any neighbor q of p, there is no neighbor r of q ($r \neq p$) that improves $\mathcal{C}$, formally $\forall q \in \mathcal{N}^{\mathcal{C}}(p), \forall r \in \mathcal{N}^{\mathcal{C}}(q) \setminus \mathcal{N}^{\mathcal{C}}(p),\ \mathcal{C}_p(r) \leq \mathcal{C}_p(q)$.

Note that, because of the partial view that a component has on the global state of the system (due to the scalability and dynamism of the system), only a heuristic algorithm can be found under these assumptions.

Theorem 1 (Local Self-Organization of LSA). *Let $\mathcal{S}$ be a system and $\mathcal{C}$ be an insensitive evaluation criterion. If $\mathcal{S}$ executes the LSA algorithm with $\mathcal{C}$, then $\mathcal{S}$ is a locally self-organizing system for any strictly monotonic local aggregation of $\mathcal{C}$.*

Module 1 Local Self-Organization Algorithm for Criteria $\mathcal{C}$ Executed by p (LSA)

```
Inputs :
    $\mathcal{C}_p$ : the evaluation criterion used by $p$;

    $\mathcal{N}^{\mathcal{C}}(p)$: $p$ neighbors for the evaluation criterion $\mathcal{C}$;

Actions :
    $\mathcal{R}$ : if $\exists q \in \mathcal{N}^{\mathcal{C}}(p), \exists r \in \mathcal{N}^{\mathcal{C}}(q) \setminus \mathcal{N}^{\mathcal{C}}(p), \mathcal{C}_p(q) \leq \mathcal{C}_p(r)$
         then $\mathcal{N}^{\mathcal{C}}(p) = \mathcal{N}^{\mathcal{C}}(p) \bigcup \{r_{max}\} \setminus q$;
                 where $r_{max} \in \mathcal{N}^{\mathcal{C}}(q)$, $\mathcal{C}_p(r_{max}) = \max_{r' \in \mathcal{N}^{\mathcal{C}}(q), \mathcal{C}_p(q) \leq \mathcal{C}_p(r')}(\mathcal{C}_p(r'))$
```

Proof. Let p be a processor in the system executing the LSA algorithm. Assume that $\mathcal{S}$ does not locally self-organize in the neighborhood of p. That is, there is an execution of $\mathcal{S}$, say e, that does not include a p-stable configuration.

Assume first that e is a static execution (i.e., no connection/disconnection action is executed during e). Let c be the first configuration in e. By assumption of the proof, c is not p-stable. Thus there is a neighbor of p, say q, that has itself a neighbor improving the evaluation criterion. Hence, rule $\mathcal{R}$ (Module 1) can be applied which makes r replacing q in the neighbors table of p. By applying the assumption of the proof again, the obtained configuration is not stable, hence there is at least one neighbor of p which has a neighbor which improves the evaluation criteria. Since the evaluation criteria is bounded and since the replacement of a neighbor is done only if there is a neighbor at distance 2 which strictly improves the evaluation criteria, then either the system converges to a configuration c_{end} where the evaluation criteria reaches its maximum for some neighbors of p, or the evaluation criterion cannot be improved.

In other words, for each node q neighbor of p we can exhibit a finite maximal string:

$$\mathcal{C}_p(q_0) < \mathcal{C}_p(q_1) < \ldots < \mathcal{C}_p(q_m)$$

where q_0 is the node q and q_i, $i = \overline{1, m}$ are the nodes which will successively replace the initial node q. Let c_{end} be the configuration where the node q_m is added to the neighbors table of p. In c_{end} the value of $C_p(q_m)$ is maximal hence, either c_{end} is stable, or no neighbor of q_m improves the evaluation criteria. Thus c_{end} is stable. Consequently, there exists a configuration in e, namely c_{end}, that is p-stable.

Assume now that the execution e is dynamic, hence the system size and topology may be modified by nodes connection and disconnection. Assume that node p joins the system. This case is similar to the previous one, where p executes the rule $\mathcal{R}$ of Module 1 a finite number of times until it reaches a p-stable configuration.

Now, let us study the case where the system is in a p-stable configuration and, due to the connection/disconnections the p neighbors set changes. That is, in the p neighbors set a node r appears, and the new node r is improving the criterion. Once p is aware of the new configuration of it neighbor it restarts the convergence period by applying rule $\mathcal{R}$. The system reaches in a finite number of steps a p-stable configuration.

3.2 Self-Organization Through Local Self-Organization

As previously said, self-organization strongly relies on the local self-organization property, as well as on the effect of connection/disconnection actions and data modifications on the system. According to this effect, the system guarantees different levels of self-organization, namely, from weak to strong self-organization. Before defining these properties, we introduce the notion of *global evaluation criterion*, denoted in the following γ. The global evaluation criterion evaluates the global organization of the system at a given configuration. More precisely, the global evaluation criterion is the aggregate of all local criteria. For instance, if the evaluation criterion is logical proximity (i.e., the closer a process, the higher the evaluation criterion), then optimizing the global evaluation criterion γ will result in all processes being connected to nearby processes.

Let $\mathcal{C}_p$ be an evaluation criterion and let γ_p be its local aggregation for any p process in the system. In the sequel we focus only on global evaluation criteria γ that exhibit the following property :

$$\forall f, \forall c_1, c_2 \in f,\ \gamma(c_1) < \gamma(c_2) \text{ if } \exists p,\ \gamma_p(c_1, \mathcal{C}_p) < \gamma_p(c_2, \mathcal{C}_p) \text{ and } \forall t \neq p,\ \gamma_t(c_1, \mathcal{C}_t) \leq \gamma_t(c_2, \mathcal{C}_t)$$

Intuitively, the increase of the value of a local criterion will determine the increase of the global criterion if the other local criteria increase their values or remain constant. An example of criterion that meets such a requirement is the union/intersection of local criteria. Namely, γ is the sum of a local aggregation criterion γ: $\gamma(c) = \sum_{p \in \mathcal{S}} \gamma_p(c)$.

The weak self-organization is defined in terms of two properties. The *weak liveness* property says that for each static fragment f_i, either (1) f_i is stable, or (2) there exists some fragment f_j, in the future of f_i, during which the global evaluation criteria strictly improves (see Fig. 1). The *safety* property requires that the global evaluation criteria never decreases during a static fragment. Formally, we have:

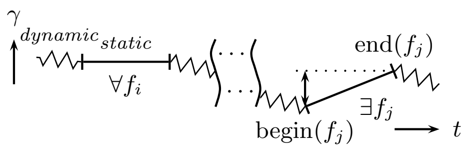

Fig. 1. Illustration of the Weak liveness property.

Definition 3 (Weak Self-Organization). *Let $\mathcal{S}$ be a system and γ be a global evaluation criterion defined on the configurations of $\mathcal{S}$. A system is weakly self-organizing for γ if the following two properties hold (recall that $(f_0, \ldots, f_i, \ldots)$ stand for static fragments):*

Weak Liveness Property:

$$\forall e = (f_0, \ldots, f_i, \ldots, f_j, \ldots), \forall f_i \in e, \exists f_j \in e, j \geq i : \; \gamma(end(f_j)) > \gamma(begin(f_j)) \\ \text{or } \forall p \in \mathcal{S}, begin(f_j) \text{ is } p\text{-stable}$$

Safety Property: $\forall e = (f_0, \ldots, f, \ldots), \forall f \in e : \gamma(end(f)) \geq \gamma(begin(f))$

The following theorem gives a sufficient condition to build a weakly self-organizing system:

Theorem 2 (Weak Self-organization). *Let $\mathcal{S}$ be a system and γ be an insensitive global evaluation criterion. $\mathcal{S}$ is weakly self-organizing for γ if for any process p, $\mathcal{S}$ locally self-organizes in p's neighborhood.*

Proof. Let e be an execution of $\mathcal{S}$.

Safety proof Let f be a static fragment in e. Since $\mathcal{S}$ is locally self-organizing then for any p node in the system there are two situations: (1) p is executing some actions in f hence $\gamma_p(\text{begin}(f)) < \gamma_p(\text{end}(f))$ or (2) p does not execute any action and in this case $\gamma_p(\text{begin}(f)) \leq \gamma_p(\text{end}(f))$. Overall, $\gamma(\text{end}(f)) \geq \gamma(\text{begin}(f))$.

Weak liveness proof Let p be a process. Let f_i be an arbitrary static fragment in e. $\mathcal{S}$ is self-organizing hence there is a static fragment f_j, $i \leq j$ in e such that p executes self-organizing actions in f_j hence $C_p(\text{begin}(f_j)) < C_p(\text{end}(f_j))$. Overall, for any f_i there is a fragment f_j such that $\gamma(\text{end}(f)) > \gamma(\text{begin}(f))$.

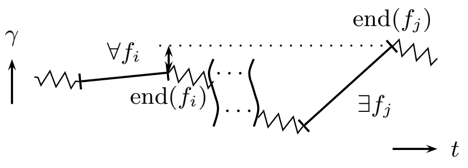

Fig. 2. Illustration of the liveness property.

The weak self-organization definition applies to static fragments. Nothing is guaranteed during dynamic ones (i.e., fragments in which connections / disconnections occur or data are modified). For example, Pastry self-organization protocol may cause the creation of multiple, isolated Pastry overlay networks during periods of IP routing failures. Because Pastry relies almost exclusively on information exchange within the overlay network to self-organize, such isolated overlays may persist after full IP connectivity resumes [7].

The following definition proposes a characterization of the system during both static and dynamic fragments. This definition is characterized by the safety property as defined above and a liveness property. This property says that either (1) infinitely often, there are static fragments during which the knowledge of the system enriches (see Fig. 2), or (2) all the processes have reached a stable state.

Definition 4 (Self-Organization). *Let $\mathcal{S}$ be a system and γ be a global evaluation criterion defined on the configurations of $\mathcal{S}$. A system is* self-organizing *for γ if both safety (defined here above) and liveness hold, with the liveness property defined as follows:*

Liveness Property:

$$\forall e = (f_0, \ldots, f_i, \ldots, f_j, \ldots), \forall f_i \in e, \exists f_j \in e, \quad j \geq i : \gamma(end(f_j)) > \gamma(end(f_i))$$
$$\text{or } \forall p \in \mathcal{S}, begin(f_j) \text{ is } p\text{-stable}$$

Theorem 3 (Self-organization). *Let $\mathcal{S}$ be a locally self-organizing system. If for any execution $e = (f_0, \ldots, f_i, \ldots, f_j, \ldots)$ of $\mathcal{S}$ and for all static fragments f_i, f_{i+1} in e, $\gamma(end(f_i)) \leq \gamma(begin(f_{i+1}))$ then $\mathcal{S}$ is self-organizing.*

Proof. From the local self-organization of $\mathcal{S}$, $\forall f_i$, $\exists f_j$ such that $\gamma(\text{begin}(f_j)) < \gamma(\text{end}(f_j))$. Using the hypothesis, $\gamma(\text{begin}(f_i)) \leq \gamma(\text{end}(f_i)) \leq \gamma(\text{begin}(f_{i+1})) \ldots \leq \gamma(\text{begin}(f_j)) < \gamma(\text{end}(f_j))$. Thus, $\mathcal{S}$ is self-organizing.

Note that neither the weak nor the liveness properties forbid processes to reset their neighbors lists after each connection/disconnection. To prevent the system from "collapsing" during dynamic fragments, we need to specify a stronger property guaranteeing that for all the processes whose neighborhood has not changed, information is maintained. Specifically, this ensures the existence of a non-empty group of processes for which local information has been maintained between the end of a static fragment and the beginning of the subsequent one. We can see this group of processes as the kernel of the system. More precisely, given two successive configurations c_i and c_{i+1} with their associated graphs G_i and G_{i+1}, the static common core of G_i and G_{i+1} is the sub-graph common to G_i and G_{i+1} minus all nodes for which the neighborhood has changed. Formally, let $G1$ and $G2$ be two graphs, and $\Gamma_{G_i}(a)$ the set of neighbors of a in G_i. We define the topological static common core of $(G1, G2)$ as:

Notation 1 (Topological Kernel) $KerT(G1, G2) = G1 \cap G2 \setminus \{a : \Gamma_{G1}(a) \neq \Gamma_{G2}(a)\}$

Since we study systems where the self-organization may be data-oriented (typically the peer-to-peer systems), we propose a data oriented definition of the static core of the system. That is, given two successive configurations c_i and c_{i+1}, the data static common core of c_i, c_{i+1} is:

Notation 2 (Data Kernel) $KerD(c_i, c_{i+1}) = D_i \cap D_{i+1}$*, where* D_i *is the system data in* c_i.

This leads to the following property :

Definition 5 (Kernel Preservation). *Let $\mathcal{S}$ be a system and γ be a global evaluation criterion defined on the configurations of $\mathcal{S}$. Let $e = (f_0, \ldots, f_i, f_{i+1}, \ldots)$ be an execution of $\mathcal{S}$ and let $K_i = Ker^*(end(f_i), begin(f_{i+1}))$ (where Ker^* denotes either $KerT$ or $KerD$). $\mathcal{S}$ verifies the kernel preservation property for γ if the following property holds:*

Kernel Safety: $\forall i, \gamma(Proj_{|K_i}(end(f_i))) \leq \gamma(Proj_{|K_i}(begin(f_{i+1})))$ *where $Proj_{|K_i}(c)$ is the sub-configuration of c corresponding to the kernel K_i.*

This leads to a stronger version of self-organization defined as follows:

Definition 6 (Strong Self-Organization). *Let $\mathcal{S}$ be a system and γ be a global evaluation criterion defined on the configurations of $\mathcal{S}$. $\mathcal{S}$ is* strongly self-organizing *for γ if it is self-organizing and it verifies the kernel preservation property defined here above.*

The concept of self-organization can be easily extended to a finite set of criteria. In the following we show that when criteria are not interfering, i.e., when they are independent, then one can build a self-organizing system for a complex criterion by using simple criteria as building blocks. Using the previous example where the local evaluation criterion was proximity, a second global evaluation criterion is needed to decrease the number of hops of a lookup application. For instance, we may want to use a few long links to reduce the lookup length.

Definition 7 (Independent Criteria). *Let $\mathcal{S}$ be a system and let γ_1 and γ_2 be two global criteria defined on the configurations of $\mathcal{S}$. Let c be a configuration of S and sc and sc' the sub-configurations of c spanned by γ_1 and γ_2. γ_1 and γ_2 are independent with respect to c if $sc \neq sc'$. γ_1 and γ_2 are independent with respect to $\mathcal{S}$ if for any configuration c in $\mathcal{S}$, γ_1 and γ_2 are independent with respect to c.*

Definition 8 (Monotonic Composition). *Let S be a system and let $\gamma_i \in I$ a set of criteria on the S configurations. $\gamma = \times_{i \in I}\gamma_i$ is a monotonic composition of the criteria $\gamma_i, i \in I$ if the following property is verified: $\forall c_1, c_2, \gamma(c_1) < \gamma(c_2)$ iff $\exists i \gamma_i(c_1) < \gamma_i(c_2)$ and $\forall j \neq i \in I, \gamma_j(c_1) \leq \gamma_j(c_2)$.*

Theorem 4 (Multi-criteria Self-orgnization). *Let $\mathcal{S}$ be a system and let $\gamma_1 \ldots \gamma_m$ be a set of independent evaluation criteria. If S is weakly, resp. strongly, self-organizing for each γ_i, $i \in [1..m]$ then S is weakly, resp. strongly, self-organizing for $\gamma_1 \times \ldots \times \gamma_m$.*

Proof. Let e be a configuration of S and let e_i be the projection of e on the sub-configurations modified by γ_i. Since, S is self-organizing with respect to γ_i then e_i is self-organizing with respect to γ_i.

Safety proof Let f be a static fragment in e and let f_i be the projection of f on the sub-configurations spanned by γ_i. From the hypothesis, $\gamma_i(\text{begin}(f_i)) \leq \gamma_i(\text{end}(f_i))\ \forall i$ hence $\gamma_i(\text{begin}(f)) \leq \gamma_i(\text{end}(f))$. So, $\gamma(\text{begin}(f)) \leq \gamma(\text{end}(f))$.

Weak liveness proof Let f_i be a fragment. There is f_j and γ_k such that $\gamma_k(\text{begin}(f_j)) < \gamma_k(\text{end}(f_j))$. Using the safety for all $\gamma_j, j \neq k$ it follows $\gamma(\text{begin}(f_j)) < \gamma(\text{end}(f_j))$.

Overall, $\mathcal{S}$ is weak self-stabilizing for γ. The proof for strong self-organization follows using a similar reasoning.

Theorem 5 (Self-organization Hierarchy). *Weak self-organization $\subset$ self-organization $\subset$ strong self-organization*

Proof. Straigtforward from the definitions.

4 Case Study : Self-organization of CAN

We now prove the self-organization of CAN. CAN [13] is a scalable content-addressable network, the principle of which is to use a single namespace—the d-dimensional torus $[0,1]^d$—for both data and nodes. Data and CAN nodes are assigned unique names within this namespace, and each node is responsible for a volume surrounding its identifier in the torus. The distributed algorithm executed on a node arrival or departure ensures that the complete torus volume is partitioned between all participating CAN nodes.

These algorithms are crucial for the self-organization of the system, since the topology of CAN changes only when nodes enter or leave the system. In the following, we show how these protocols fit into our self-organization framework. Let us consider the following evaluation criterion :

$$C_p^{CAN}(q) = \frac{1}{1 + dist(p,q)}, \text{where } dist \text{ is the cartesian distance in the torus}$$

Theorem 6. *The CAN system is a weakly self-organizing system using the C_p^{CAN} criterion.*

Proof. We first show that the CAN protocols for node insertion and node removal perform actions that leave the system in a p-stable configuration (weak self organization), then show that, during unstable periods, the system exhibits the kernel preservation property.

Node removal When a node leaves the system, its previous neighbors' evaluation criteria decrease, since the distance to the leaving node is now set to ∞. As we are only concerned with fragments in which no disconnection can occur, let us consider actions taken by the protocol following the leaving of the node p. Just after the departure of p, every neighbor of p saw a decrease in its evaluation function, and starts to look for a new neighbor. The algorithm used by CAN [14, 13] is designed in such a way that the newly chosen neighbor is optimal with respect to the Cartesian distance. Hence, in the fragment *following* the leaving of p, the criterion for every neighbor of p increases. Once every previous neighbor of the leaving node is finished with the protocol, the topology of CAN does not change unless a connection or another disconnection occur. Hence, the departure protocol leaves the system in a p-stable configuration.

Node insertion The insertion of a node is a two-step operation. In the first step, the node p that wants to join the system computes an id, which is a point in the d-torus, then gets the IP address of some CAN node q_0. The second step is the actual insertion: (1) q_0 sends a message to the node q_1 responsible for the volume containing the id computed by p, then (2) p contacts q_1 which, in turn, contacts its neighbors and splits its volume in order to maximize the uniform distribution of nodes within the torus, and finally (3) p enters the system with a volume defined by its id and by q_1 and its neighbors.

The key point here is that, for any node r in the torus, when a new node p is inserted in CAN, it becomes a neighbor of r only if p is closer to r than one of r's previous neighbors. Hence, the Cartesian distance from r to its neighbors is either the same or reduced, when compared to the situation before the insertion: the evaluation criterion for every node in the system is improved by an insertion, thus CAN is weakly self-organizing.

Note that, when CAN experiences only connections, the evaluation criterion in the common core (see Definition 5: nodes that keep their neighborhood intact) is unchanged, while its value increases for the new connected nodes and their neighborhood. We can conclude that CAN, in the presence of connections, is a strong self-organizing system with respect to the C^{CAN} criterion (see Definition 6).

Multi-layering in CAN Another feature of CAN is its ability to support multiple *realities* [14, 13]: several coordinate spaces can be used in parallel, in a layered form. For example, in a scenario where three different realities coexist in CAN, every node of the system has three different coordinates and, correspondingly, three lists of neighbors, one for each layer. These realities are completely independent, and hence Theorem 4 can be used to show that multi-realities CAN is an example of a multi-criteria self-organizing system.

5 Conclusion & Open Problems

In this paper, we have proposed a framework for proving the self-organizing properties of dynamic systems. Self-organization is a key feature for the newly emerging dynamic networks (peer-to-peer, ad-hoc, robot or sensor networks). Our framework includes formal definitions for self-organization, altogether with sufficient conditions for proving the self-organization of a dynamic system. We have illustrated our theory by proving the self-organization of two p2p overlays: Pastry and CAN.

We have also provided a generic algorithm that ensures the self-organization of a system with respect to a given input criterion. Our algorithm is based on the greedy technique, and relies solely on the local knowledge provided by the direct neighborhood of each process. This algorithm can be used as building-block in the construction of any self-organized DHT-based or unstructured peer-to-peer system.

Several problems are left open for future investigation. The first one is the design of a probabilistic extension to our model. This study is motivated by the fact that connection/disconnection actions are well-modeled by probabilistic laws. Essentially, the liveness property could be redefined using the Markov chains model for probabilistic dynamic I/O automata. Moreover, since our generic algorithm for self-organization uses a greedy deterministic strategy, it may reach just a local maximum for the global criterion. Adding randomized choices could be a way to converge (with high probability) to a global maximum.

Another interesting research direction is to prove or refute our conjecture that the selfish self-organizing generic strategy (Algorithm LSA) is optimal among all the self-organizing local strategies. Games and economic mechanisms theories are rich in tools adequate to this study.

We also intend to extend our framework towards a unified theory of the self* (self-healing, self-configuration, self-reconfiguration, self-repairing) properties of dynamic systems. To this end, we need to extend our case study to other dynamic systems like robots networks and large scale ad-hoc or sensor networks, that may offer complementary insides for understanding the rules that govern complex adaptive systems.

Finally, we would like to study the relationship between the self-organization and super-stabilization [6]. Note that CAN and Pastry are not self-stabilizing or super-stabilizing (direct consequence of Theorem 1, [1]). We conjecture that self-organization and super-stabilization are two complementary notions.

Acknowledgment

We thank Lisa Higham for her valuable comments on an earlier version of this paper. Her suggestions are at the origin of the Case Study section. We thank also Joffroy Beauquier for pointing us the super-stabilization work [6].

References

1. E. Anceaume, X. Défago, M. Gradinariu, and M. Roy. Towards a theory of self-organization. Technical Report 1694, IRISA, 2005.
2. P. Attie and N. Lynch. Dynamic input/output automata: a formal model for dynamic systems. In *Proc. of the 20st Annual ACM Symposium on Principles of Distributed Computing (PODC'01)*, pages 314–316, July 2001.
3. O. Babaoglu, H. Meling, and Montresor A. Anthill: A framework for the developments of agent-based peer-to-peer systems. *ICDCS 2002*, 2002.
4. S. Capkun, L. Buttyan, and J. P. Hubaux. Self-organized public-key management for mobile ad-hoc networks. *Transactions on Mobile Computing*, January-March 2003.

5. S. Dolev. *Self-Stabilization.* The MIT Press, 2000.
6. Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997.
7. P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, May 2001.
8. K. Fujibayashi, S. Murata, K. Sugawara, and M. Yamamura. Self-organizing formation algorithm for active elements. *SRDS'02*, pages 416–422, October 2002.
9. L. Garcés-Erice, E. W. Biersack, and P. Felber. Multi+: Building topology-aware overlay multicast trees. In *QofIS*, pages 11–20, 2004.
10. G. Kan. *Harnessing the benefits of a disruptive technology.* O'Reilley & Associates, March 2001.
11. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, R. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.
12. Alberto Montresor, Márk Jelasity, and Özalp Babaoğlu. Robust aggregation protocols for large-scale overlay networks. In *Proc. DSN*, pages 19–28, June 2004.
13. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM'01*, pages 161–172. ACM press, 2001.
14. Sylvia Paul Ratnasamy. *A Scalable Content-Addressable Network.* PhD thesis, University of California at Berkeley, 2002.
15. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 4th IFIP/ACM Middleware Conference (Middleware '01)*, pages 329–350, 2001.
16. A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 188–201, 2001.
17. K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in p2p systems. *Proc. Infocom'03*, 2003.
18. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIG/COMM*, pages 149–160, aug 2001.
19. I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: formation of geometric paterns. *SIAM Journal of Computing*, 28:1347–1363, 1999.
20. J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, pages 171–180, 2000.
21. H. Zhang and A. Arora. Gs3 : Scalable self-configuration and self-healing in wireless networks. *Proc. of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC'02)*, pages 58–67, 2002.
22. B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.