Bio-inspired Self-organization Methods and Models for Software Development

Daniel Joseph Dubois

Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
dubois@elet.polimi.it

Abstract. The current research trends in Software Engineering are focusing on the development of new techniques to deal intelligently and efficiently with the design of systems that are able to evolve overtime and adapt to rapid changes of their requirements. In particular, the field of Autonomic Computing has been created to study these types of systems with the ultimate aim to create systems that are able to self-configure, self-optimize, self-heal and self-protect without any external intervention. What we analyze in this thesis is a set of the most common bio-inspired principles, methods, and models that may be applied to these systems. In particular we want to propose a way to apply them to develop or adapt self-organization algorithms to real evolving systems and a methodology to validate them using either formal proofs or extensive simulations. The final outcome of the work will include a real application of such methodologies in different scenarios.

1 Introduction

The current research trends in Software Engineering are focusing on the development of new techniques to deal intelligently and efficiently with the design of systems that are able to evolve overtime and adapt to rapid changes of their requirements. In particular, the field of Autonomic Computing [1] has been created to study these types of systems with the ultimate aim to create systems that are able to self-configure, self-optimize, self-heal and self-protect without any external intervention. More recently the Autonomic Computing area has started very strong interactions with different related fields such as Pervasive Computing [2], Situational Computing [3], etc: all these fields have in common the fact that they need to be open in order to follow the changes that may happen in the external world [4].

The purpose of this thesis is to focus on bio-inspired self-organization algorithms and to present our experience in exploiting them in the development of distributed software architectures. These types of algorithms often need to be tailored to the real system in which they are executed. Finding the correct tailoring is not always trivial since it requires extensive modeling and simulations in the presence of an execution context that may be continuously changing.

After an initial analysis of the state of the art in developing self-adaptive and self-organizing systems we discuss the most common principles that may be encountered in bio-inspired self-organization, we then want to propose innovative techniques that can be used to apply these principles and algorithms to real systems, with a particular emphasis on the methodology that may be use to validate them. One of the future outcomes is to provide methods to analyze these approaches both in a formal way, using model checking and other mathematical validation techniques, and an experimental way using extensive simulations and other statistical techniques that may be applied to the simulation results.

The organization of this paper is as follows. Section 2 presents a state of the art analysis of self-organization in software engineering, a classification of the most recurring bio-inspired self-organization principles, and examples of algorithms that use them. In Section 3 we discuss our existing ongoing study on the issues that arise when adapting bio-inspired self-organization algorithms to real software systems and the current approaches we are using to validate them. Finally, Section 4 shows the plan to develop the research discussed so far.

2 Background information and related work

In this section we discuss the existing background and ongoing work in the area of system self-organization in software engineering with particular emphasis on the case of bio-inspired self-organization.

2.1 Self-organizing systems

To reduce the dependency on manual interventions many different software architectures have been proposed. IBM proposed a generic architecture [5] of an autonomic system composed by two entities: a manager and some managed resources. In this approach the manager communicates with the resources through a sensor/actuator mechanism and the decision is elaborated using the so-called MAPE cycle in which the manager *Monitors* the sensors, *Analyzes* the collected data, *Plans* an action, and then *Executes* it using the actuators.

In addition to this preliminary research there are other different approaches such as the model-based adaptation of Garlan [6] in which the architecture of the system may change at runtime depending on what is being monitored, and a more recent work from Kramer&Magee [7], where they try to exploit the analogies between robotic systems and self-managed systems.

Other self-adapting architectures that use different approaches are: Autonomia [8], which uses a two layer approach, where the first contains the execution environment and the other manages the resources; AutoMate [9] which has a multi-layered architecture optimized for scalable environments such as decentralized middleware and peer-to-peer applications; SelfLet [10] and CASCADAS [11] approaches, which have both been designed to have runtime changing goals, plans, rules, services and a behavior modeled as a final state machine that may change overtime.

While the above approaches work at the level of the application by suggesting a way to offer it some self-* properties, in literature there are other types of approaches that are specifically focusing on the middleware-level: they show self-adapting capabilities for optimizing some of middleware properties/metrics. For instance, in distributed-dispatcher publish-subscribe systems some approaches are presented that enable the reconfiguration of the underlying topology of the distributed dispatcher to minimize some cost metrics such as the network traffic [12–14], while in peer-to-peer networks based on unstructured overlays other approaches are mainly focusing on mechanisms to search/share some information [15–17].

All the described architectures are supported at run-time by an algorithmic self-organizing logic. These algorithms may be used into different levels of existing systems. An example is the distributed monitoring mechanism presented in [18]: special architectural components called supervisors collect status data from the underlying components and decide whether to trigger or not corrective reactions.

Finally another class of self-organization algorithms, which will be the central topic of this thesis work, is composed of bio-inspired algorithms. The main characteristic of these algorithms is the fact that they are based upon a set of principles inspired by the natural world and that they provide simple solutions to solve problems that would be much harder using classical approaches. Characteristics of these algorithms are discussed in detail in the following subsection.

2.2 Bio-inspired Self-organization

Self-organization is defined as *"The spontaneous evolution of a system into an organized form in the absence of external pressures"* [19]. Forms of self-organization that come from observed phenomena of the natural world are called *bio-inspired self-organization*.

In this subsection we propose a list of the most important principles of bio-inspired self-organization and some algorithms that are based upon them.

Principles Bio-inspired Self-organization is usually based upon common principles [20] taken from the natural world that may be composed and translated into algorithms. The following is a list of some of these principles.

Noise. This principle [21] is defined as the use of some kind of perturbations that move the system away from its expected goal in the short term, but makes it possible with a certain probability to reach a better goal in the long term. In other words if the system goal is measured by a goal function that should be maximized, then the use of noise makes it possible to increase the probability to move away from the local optima of that function and then to move to the global optimum. This principle works in a similar way to what we see in genetic algorithms [22] in which the noise is added by using the mutation/crossover operations. Noise is usually present in most bio-inspired systems regardless their

design and it is particularly useful in improving the solution in optimization problems.

Emergence. This principle [23] is defined as the capability of a system composed by multiple components to reach, with a certain probability, a global goal by achieving local goals at component level that may be apparently unrelated to the global goal. Examples of emergent behaviors are very common in nature, like the phenomenon of fireflies synchronization, in which the local goal of a firefly is to modify its blinking frequency to synchronize with another firefly in its sight range, while the global goal that emerges is having a community of fireflies that blink at the same time. Creating an emergent property in a software system is not always obvious: a possible simple method may be to decompose, in a top-down way, the global problem into smaller subproblems translated into simple rules to be run by system components, however in most of the cases components rules cannot be easily derived from the global goal and, vice-versa, global goals are gradually discovered in a bottom up way from the analysis of the characteristics of local goals.

Diffusion. This principle [24] is defined as a method for communicating information among many interconnected components. According to this principle the information produced or received by a node is sent to some other nodes regardless the destination of the message. The destination nodes may be any neighbor of the sending node (flooding) or some random nodes (gossiping). The final aim of this mechanism is to increase the probability that nodes interested to that message actually receive it. Diffusion is usually used in peer-to-peer networks; possible uses are the search or synchronization of information. This communication method usually gives self-healing/fault-tolerance properties to the system since messages are usually redundant and thus the removal (or malfunction) of some system components does not prevent the correct destinations to receive the messages.

Stigmergy. This principle [25] is defined as another method of communicating information among different moving components of a system. According to this principle the information is not sent to other nodes, but it is stored in the environment. Since components are capable of moving, when they change their location (context) they may read in the new environment the information previously left from another component, use it, and possibly update it. Environmental information may be persistent or it may expire after a timeout. This phenomenon in nature may be noticed under the form of pheromone evaporation. Since this principle relies on leaving information on the environment, the failure of some components does not compromise the communication (assuming that the probability of an environment failure is negligible).

Evolution. This principle [26] is defined as the capability of the system to improve itself using a *natural selection* process among its components: the best

components tend to survive, the worst components tend to die. During this process best components may be replicated, partially mutated, and recombined with other components. A well-known class of algorithms that use this principle are the genetic algorithms [22].

Algorithms In this subsection we explain some examples of bio-inspired algorithms and frameworks presented in literature that are related to the principles discussed above.

Genetic algorithms [22] are examples of application of the evolution principle. These type of algorithms consider an initial *population* of preliminary solutions to a problem and then, using a *fitness* function, they quantify the quality of each solution. Then the worst solution are discarded and the best ones are left. To avoid local optima in the solution the best solutions are combined together and changed slightly using the crossover and mutation operations. This last operation is an application of the noise principle.

A different class of algorithms is the epidemic algorithms class. These algorithms resemble the spread of a contagious disease and rely heavily on the diffusion principle. They have been proved to work in the following classes of problems: failure detection, data aggregation among internal knowledge of system components, creation of groups, etc. An example of application of such algorithms in distributed system has been proposed by Guerraoui et al [27].

An example of emergence is provided also in a work from Saffre et al [28]. In this work each system component starts as a generic worker and then, if certain conditions are met after an interaction among components, components may decide to differentiate themselves and become more efficient in performing specific tasks, with the cost of being no longer able to execute generic tasks.

Another very popular class of bio-inspired algorithms was proposed by Dorigo [29]: the Ant Colony Optimization algorithm. This algorithm has been used to solve combinatorial optimization problems by reducing them to the generic problem of ants looking for the optimal path from their anthill to the food source. This is a typical example in which we can see all the principles applied to the same problem.

A bio-inspired study in which our research group was involved in the last years includes the solution of the components aggregation problem in dynamic networks [30, 31], in which each component, using simple interactions, is able to efficiently rewire the network to increase the probability that each component will eventually increase the number of similar components in its neighborhood.

Some of other popular examples of bio-inspired self-organization focus on the problem of reorganizing the topology of the nodes in a communication network [32, 33, 17, 34]. These approaches have in common the fact that they are able to make emergent properties appear in the network topology using different metrics to decide whether to add/remove links among nodes, moreover they rely often on the concept of noise (such as what happens in Cyclon [17] when performing the periodical shuffling of neighbors nodes).

AntHill [35] is another bio-inspired algorithm-based framework that focuses on the design of self-organizing systems that are built on top of peer-to-peer applications. In this approach each network component is called *nest* and each nest may host a *society* of simple autonomic agents called *ants*. In AntHill system requests are handled by nests: after they receive a request a population of ants is generated. Ants may communicate using the stigmergy principle by modifying their local environment that corresponds to the visited nests. These simple local interactions permit at the end the emergence of complex system-wide goals.

Stochasticity A common characteristic of bio-inspired algorithms is their stochasticity. This characteristic derives from the fact that bio-inspired principles are able to apply self-organizing properties in a probabilistic way. This means that they may be used to increase the probability to self-organize the system, but do not assure that the system actually self-organize. This may be confusing at the beginning since in classical software engineering any algorithm whose preconditions are satisfied may either work (correct behavior) or not work (wrong behavior). When we use bio-inspired algorithms our question is no longer *when they actually work*, but *what the probability that they work is*. Therefore, as long as the correct-behavior probability is respected, the fact that an algorithm does not work or that the system stays in a wrong status in presence of verified preconditions is not considered a wrong behavior, but the expected behavior. The result is that all the properties of the system (both functional and non-functional) are expressed in stochastic terms. Some of the advantages of stochasticity are the fact that the system tends to keep working in presence of temporary unexpected behavior of its components. A possible drawback is the absence of guarantees on when a specific goal is reached: this may make bio-inspired approaches less suitable in solving time-constrained problems.

3 Research Hypotheses and Directions

This research aims at finding a classification of possible bio-inspired self-organization techniques and at finding a way to systematically apply them to real system. To support this we propose a preliminary tentative of methodology, various validation techniques that according to our experience may be used to validate the effectiveness and/or the efficiency of the approach, and an example of application.

3.1 Using Bio-inspired Self-organization in Real Systems

So far, bio-inspired approaches have been studied in theory and, in some cases, developed in toy examples. In this research we are currently trying to apply them in some real systems and therefore we are trying to understand if there is any repeatable approach that we can adopt to reach this goal.

According to our experience, in some cases, the self-organization algorithms that we have presented in Section 2.2 can be simply adopted to address a specific

problem. In some other cases none of the identified algorithms can be applied as they are, but, indeed, we can rely on the adoption of some of the self-organization principles. A general idea for matching bio-inspired principles to the problems we want to solve is to classify the problems into classes that may be reduced to the typical situations in which each principle has proven to be useful. For example difficult optimization problems may take advantage of perturbations/noise, communication in presence of uncertainty and hundreds/thousands of entities requires diffusion, and so on. A more systematic approach for doing this will be part of the outcome of this thesis, however we will show in Section 3.3 how some of the principles we have identified so far can be applied to the case of the reconfiguration of a publish/subscribe architecture.

As soon as we have identified the algorithms or the principles that are most suitable to a specific case, according to any well-defined software engineering approach, we need to build a model for our system to check that it behaves as expected. This step is particularly important here because of the inherent complexity and distribution of the systems we consider, and on the consequent difficulty of building them directly without any design level check on the feasibility of the approach.

As soon as we are convinced that our model works, we can start implementing it. Even in this phase we should not be limited to a simple model-to-code approach since we have to face with non-trivial problems that are not usually faced in the theoretical definition of the algorithms and principles, and therefore could have not been captured in our model. These problems often concern synchronization among components, management of race conditions, the driving of the system toward some initial state that is suitable for the algorithm to start, the identification of the conditions under which the self-organization algorithm could start and end, and when to actually repeat an iteration of the algorithm without using too much computational time, but still achieving a good level of responsiveness. In fact, it is very common that bio-inspired self-organization algorithms are proposed and studied only as a set of rules to be executed for every iterations by abstracting away from all the details that cause the issues that we have listed above. A summary of the steps discussed above can be seen in Figure 1.

The solution to these problems depends not only on the problem/solution model itself, but also on the deployment scenario: a wrong implementation choice at this level may nullify the effectiveness of the algorithm at all. Thus, even in this phase, the role of models and analysis and simulation approaches is prominent in order to provide some level of guarantees on the quality of the implementation.

The last aspect to be taken into account has to do with the stochasticity of the self-organization approaches. We need to be aware of this and, in case we cannot tolerate it, we have to structure the system in such a way that, when it is not able to converge to the expected state in a time interval that is acceptable for the specific application we are developing, some other more predictable mechanisms take the lead and guarantee the expected convergence.

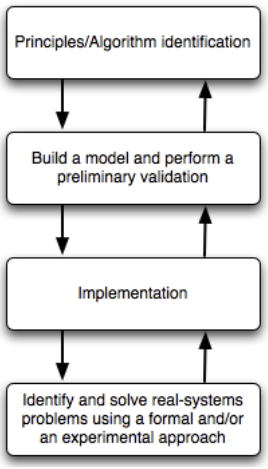


Fig. 1. General steps to develop a bio-inspired self-organization algorithm for a real system.

In the following paragraph we describe one example in which we try to show what we did to address the aforementioned issues.

3.2 Modeling and Validating Approaches

This subsection discusses the modeling and validating aspects of this research. In particular we discuss two different classes of approaches that are able to model and validate the typical systems that may take advantage of bio-inspired self-organization.

All the following approaches require that each component of the system and the interaction among different components are clearly specified. The question we want to answer after validation is whether the algorithm works or not, if it is close to the (global) optimum, and if it is resistant to unpredictable situations.

Experimental Approaches This class of approaches use the specification of the system, the constraints on its evolution, and the proposed algorithm to implement them directly on a simulation (or real) testbed in order to see how the algorithms behave in practice.

The system is usually initialized using data that makes it more similar to a real deployment scenario and let it run for tens or ever hundreds of times. This method, called Monte Carlo simulation method, uses an aggregation of the results of all these runs to perform some statistical analysis. The result of the analysis usually gives information about the confidence that the algorithm is able to reach its goal. The stability of the algorithm is usually tested by adding artificial perturbations to the system, or by using deployment environments that

are intrinsically characterized by uncertainty (such as the PlanetLab network for heavily distributed applications).

The problem of the experimental approaches is that they tend to be biased towards the initial assumptions and that they are characterized by a prediction error. Results achieved using experimental approaches are usually used to generalize the validity of an algorithm from a particular case to a more general case, however this is not true all the times and a more formal approach is needed.

Formal Approaches Formal methods to analyze this type of algorithms include the following approaches:

- *Temporal Logic models*: the system and its evolution are formally modeled using temporal logic and, using a satisfiability checker, it is possible to check some generic system properties that are general and independent of the deployment scenario.
- *System Theory and Game Theory models*: the system and its evolution are modeled mathematically using differential equations. The mathematical system can then be used to find equilibria, attractors, and other information on the evolution of the system in the long term.
- *Operations research models*: the system is expressed using a series of constraints and an objective function to be maximized. The advantage of this method is that it is easier to prove the possibility to reach the global optimum.

A disadvantage of the models above is that some stochastic information is lost since it is usually not practical to mathematically (or logically) express them.

In our research we are currently focusing on the logic models, however, due to the long runs that are required by the satisfiability checker, current models are limited to just a few components. The purpose of future research is to extend the possibility to formally analyze the system with much more components using a compositional modeling approach.

3.3 Example: Overlay Self-organization in Publish-subscribe Systems

This example shows how it is possible to use bio-inspired self-organization and the validation approaches discussed in the previous section to reduce the traffic in a distributed-broker publish-subscribe system.

Let us assume a system composed of interconnected entities called *brokers*. Each of these entities may have zero or more components connected. Each component may be connected only to one broker and it may publish messages and subscribe to messages produced by some other component. Each broker is in charge of collecting the subscriptions from its components and of forwarding them to all the other brokers. This way every broker can maintain a routing table that is used to route every published messages only to components that have previously subscribed to them (subscription-forwarding approach [36]). Another

assumption is that the publish-subscribe system allows a broker to change its neighbors at runtime.

In this setting the final goal is to minimize the number of messages that traverse the brokers network by rewiring the connections among the brokers. To achieve this goal we could not use any of the algorithms that we have identified in Section 2.2 as the problem to be tackled depends on very specific information, that is, the subscriptions that are known by each broker and the expected pattern of traffic in the network. Instead, we could adopt the emergence principle to build a utility-function based algorithm in which the maximization of such function at local level moves the whole system toward the global goal. Moreover, we have also shown that the usage of the noise principle further improves the solution.

The preliminary results are reported in [34] where we define the utility function that, given a generic node N_0 and two of its neighbors called N_1 and N_2, is able to estimate the number of saved messages after removing the link between N_0 and N_1 and adding a link between N_1 and N_2. Therefore a generic broker N_0 decides to perform the rewiring using the two neighbors that are able to maximize the utility function.

As we have said in Section 3.1, implementing just the algorithm that evaluates this utility function is not enough, but we need to choose an interval between iterations and a proper locking mechanism. A possible implementation of an algorithm iteration that maximized the utility function is shown in the sequence diagram of Figure 2.

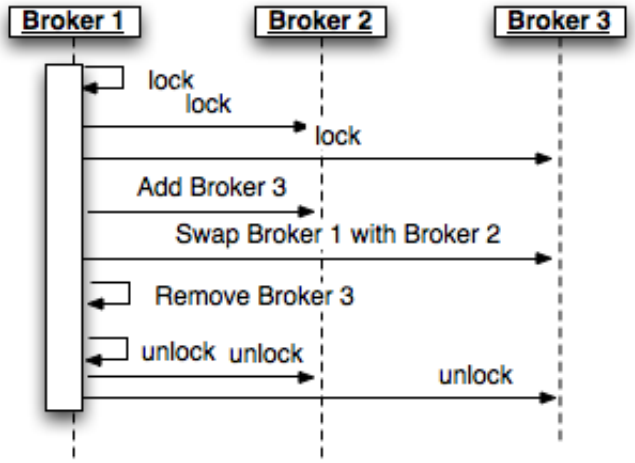


Fig. 2. Sequence diagram of a rewiring iteration that maximizes the utility function.

Validation In this example we used an experimental validation approach consisting in extensive simulations. The simulations have been carried out using a

custom simulator written in Java. A simulation is run at least 20 times to provide some statistical validity to the obtained results. The simulation parameters that have been considered are related to the characteristics of the environment on which a distributed pub/sub middleware is deployed.

The experiments have been executed in the following way: first of all we have performed several experiments to see if the algorithm was really able to reduce the network traffic using different simulation parameters, then we have selected the most significative experiments (called from now on *reference experiments*) and we varied one parameter at the time to see their impact on the algorithm results.

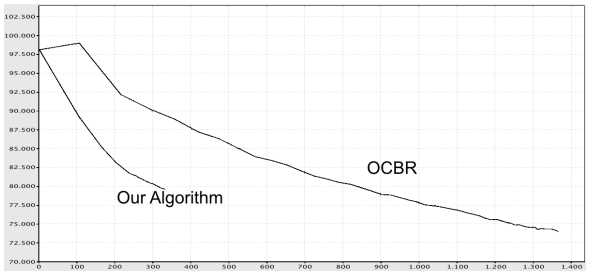
Reference experiments. The chart in Figure 3a shows the reference experiment of our algorithm and a competing approach called OCBR [14]. The lines represent the average values of the network traffic. The simulation parameters we have used are: 500 brokers, 100% of the brokers have subscribing components, 10% of the brokers have subscribing publishers, 10 publications per publisher, 1 subscription per subscriber, scale-free topology, 10 types of messages, subscriptions match 3 types of messages, 10% of algorithm noise (in terms of iterations that may not contribute to the algorithm convergence).

From the comparison in Figure 3a we can see that the heuristic of our algorithm is able to reduce the traffic with a much lower number of rewirings than the other approach in the same amount of time steps.

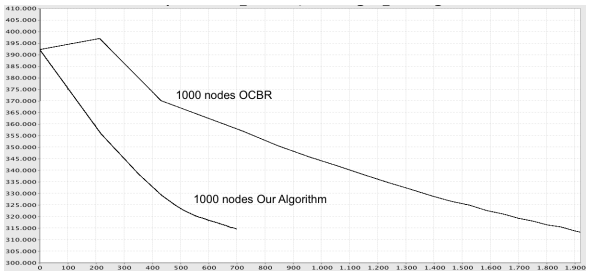
Experiments on scalability. In Figure 3b we can see that if we double the total number of brokers, we have a strong increase in network traffic due to the more complex routing of the messages, however both algorithms show to scale well and therefore to be able to reduce the traffic with a similar percentage with respect to the reference case (see Fig 3a).

Experiments on robustness. Figure 3c shows that if before every algorithm iteration 5% of the nodes is replaced by new nodes, then the proposed algorithm is still able to reduce the traffic (even if in a slower way compared to the reference experiment), while the OCBR algorithm does not have the time to reach its convergence and therefore shows during all the simulation its initial behavior of increasing the traffic. This is the typical situation in which fast (although suboptimal) convergence is fundamental in presence of uncertainty and dynamism.

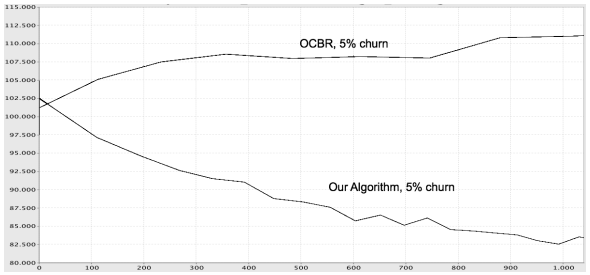
Using noise to refine the utility function. The easiest way to find a refinement for our solution is to try to add noise to the evaluation of the utility function: instead of choosing neighbors that maximize it, we relax this constraint by allowing possible rewirings that increase slightly the total number of messages that traverse that broker. Using this approach the initial iterations of the algorithm are slowed down, but, after a while, we escape the local optima and the global solution becomes better than the one obtained using the approach without noise. The results of this experiment are shown in Figure 3d. What we have learned from this last experiment is that the best solution is to have a tradeoff between



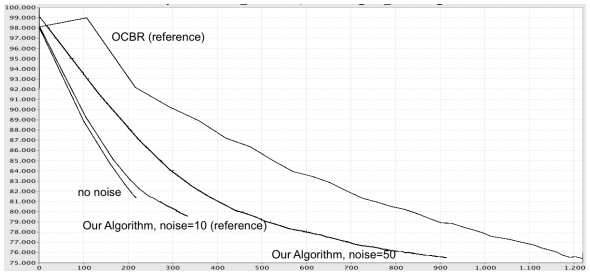
(a) Reference experiment.



(b) Varying network size (scalability).



(c) Effects of node churn (robustness).



(d) Effects of different percentages of algorithm noise.

Fig. 3. Experiments. x-axis contains the number of rewirings, y-axis contains the number of exchanged messages.

convergence rate (high with a lower noise) and effectiveness in terms of saved messages (high with a higher noise).

4 Development and Evaluation Plan

In the research activity we have carried out till know we have studied the problem of applying bio-inspired self-organization techniques to real systems. We have started this study surveying different principles that describe natural self-organizing phenomena and we have presented some existing self-organizing classes of algorithms that exploit the above principles. We have also seen that existing bio-inspired algorithms tend to be presented through toy-models that are often difficult to bring as part of a real system implementation.

The current plan to develop the preliminary research ideas presented in the previous section is to understand whether it is possible to find a common approach to actually use these types of algorithms and to deal with some of the implementation issues that may arise. So far we have learned that the process of solving a problem with a self-organization algorithm is not trivial and requires extensive reasoning about which algorithms/principles to use and especially how to apply them. Differences in the real execution environments have a heavy impact on how the algorithms should be calibrated.

The planned development direction of this work is to extend the proposed idea to a more complete methodology consisting in a complete list of steps and design patterns that may be used to apply bio-inspired principles to real systems in a more systematic way. For simplicity we divide this plan into four distinct phases.

4.1 Propose new algorithms and improve existing ones

In this part of the research plan we want to extend and improve the algorithms that have been found so far. A possible future outcome in this direction is – for example – to find a heuristic to eliminate completely the need for self-adapting the algorithm parameters (such as the interval between different iterations in the overlay self-organization example) without relying on extensive simulations on the real environment in which the algorithms would be used.

4.2 Definition of problem classes and of the appropriate solutions

The bio-inspired principles that have been discussed previously have been accompanied with some examples of existing algorithms that take advantage of that. However, as far as we know, there are not works that associate principles to their problem class in a more concrete way. For *concrete* we mean that the association is not limited to an abstract and generic statement such as "noise is used to improve the solution of optimization problems", but it also considers the noise type and the amount of noise that should be added. This last point

is generally stricly problem-dependent. The effort here is to consider some special cases that are enough representative of entire problem-classes, and finally to perform an extensive study just on them.

4.3 Mapping problems/solutions to real case studies

This third point of the research plan aims at moving to an additional level of concreteness. Until now it was not easy to exploit these techniques in existing case studies, especially because architectures characterized by centralization, replication and monitoring are easier to formalize, they are more predictable, and less affected by stochasticity. However, as we have said in the introduction section, these architectures are more difficult to be maintained once the system complexity (in term of heterogeneity and number of components) increases. The goal of this part of the research plan is to map at least each defined problem class to a case study/running example in which we provide a bio-inspired solution to the problem.

4.4 Comparing the results to traditional approaches

This last point of the research plan aims to investigate whether a software engineer should prefer using a bio-inspired approach or not. This problem can be reduced by discussing advantages and disadvantages of using our approaches and more traditional approaches to the same problems. This discussion can be carried out for example by making a direct comparison of the simulation results where we show how some different metrics change. We expect from this research that there will be problem classes and situations that are best managed by a bio-inspired self-organization algorithm and others that would continue to work better using existing techniques.

Acknowledgements

This research has been partially funded by the European Commission, Programme IDEAS-ERC (Project 227977-SMScom), FET (Project CASCADAS IST-027807), and FP7 (NoE S-Cube).

I would like to thank my thesis advisor Elisabetta Di Nitto for her support in this thesis work and also Raffaela Mirandola for insightful discussions on the topic.

References

1. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
2. Waldrop, M.: Pervasive computing - an overview of the concept and exploration of the public policy implications (March 2003)

3. C B Anagnostopoulos, Y Ntarladimas, S.H.: Situational computing: An innovative architecture with imprecise reasoning. Journal of Systems and Software **80**(12) (2007) 1993–2014
4. Baresi, L., Nitto, E.D., Ghezzi, C.: Toward open-world software: Issue and challenges. Computer **39**(10) (2006) 36–43
5. : IBM Autonomic Computing Toolkit - User's Guide. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/autonomic/books/fpu3mst.pdf`
6. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: WOSS '02: Proceedings of the first workshop on Self-healing systems, New York, NY, USA, ACM (2002) 27–32
7. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. Future of Software Engineering **0** (2007) 259–268
8. Hariri, X.D., Xue, S.L., Chen, H., Zhang, M., Pavuluri, S., Rao, S.: Autonomia: an autonomic computing environment. In: IEEE International Performance, Computing, and Communications Conference, 2003. (2003)
9. Parashar, M., Liu, H., Li, Z., Matossian, V., Schmidt, C., Zhang, G., Hariri, S.: Automate: Enabling autonomic applications on the grid. Cluster Computing **9**(2) (2006) 161–174
10. Devescovi, D., Di Nitto, E., Dubois, D.J., Mirandola, R.: Self-Organization Algorithms for Autonomic Systems in the SelfLet Approach. In: Autonomics, ICST (2007)
11. Hoefig, E., Wuest, B., Benko, B.K., Mannella, A., Mamei, M., Di Nitto, E.: On concepts for autonomic communication elements. In: International Workshop on Modelling Autonomic Communications. (2006)
12. Baldoni, R., Beraldi, R., Querzoni, L., Virgillito, A.: Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. Comput. J. **50**(4) (2007) 444–459
13. Jaeger, M.A., Parzyjegla, H., Mühl, G., Herrmann, K.: Self-organizing broker topologies for publish/subscribe systems. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, New York, NY, USA, ACM (2007) 543–550
14. Migliavacca, M., Cugola, G.: Adapting publish-subscribe routing to traffic demands. In: DEBS '07: Proceedings of the 2007 inaugural International conference on Distributed event-based systems, New York, NY, USA, ACM (2007) 91–96
15. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM (2001) 149–160
16. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Lecture Notes in Computer Science. (2001) 329–350
17. Voulgaris, S., Gavidia, D., van Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. J. Network Syst. Manage. **13**(2) (2005)
18. Baresi, L., Guinea, S., Tamburrelli, G.: Towards decentralized self-adaptive component-based systems. In: SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, New York, NY, USA, ACM (2008) 57–64
19. Heylighen, F., Gershenson, C.: Information systems, may/june 2003. the meaning of self-organization in computing

20. Babaoglu, Ö., Canright, G., Deutsch, A., Caro, G.A.D., Ducatelle, F., Gambardella, L.M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology for distributed computing. ACM Trans. Auton. Adapt. Syst. **1**(1) (2006) 26–66
21. Nicolis, S., al.: Optimality of collective choices: a stochastic approach. Bulletin of Mathematical Biology (2003) 65, 795–808
22. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
23. Holland, J.H.: Emergence: from chaos to order. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1998)
24. Datta, A., Quarteroni, S., Aberer, K., eds.: Autonomous Gossiping: A Self-Organizing Epidemic Algorithm for Selective Information Dissemination in Wireless Mobile Ad-Hoc Networks. In Datta, A., Quarteroni, S., Aberer, K., eds.: Semantics of a Networked World. Volume 3226 of LNCS., Boston, MA, USA, Springer Berlin / Heidelberg (2004)
25. Beckers, R., Holland, O., Deneubourg, J.: From local actions to global tasks: stigmergy and collective robotics. In: ALIFE IV, Brooks & P. Maes, MIT Press, Cambridge (Mass) (1994)
26. Fogel, D.B.: What is evolutionary computation? IEEE Spectr. **37**(2) (2000) 26–32
27. Eugster, P.T., Guerraoui, R., Kermarrec, A.M., Massoulieacute;, L.: Epidemic information dissemination in distributed systems. Computer **37**(5) (2004) 60–67
28. Saffre, F., Halloy, J., Shackleton, M., Deneubourg, J.L.: Self-organized service orchestration through collective differentiation. IEEE Trans Syst Man Cybern B Cybern **36**(6) (2006) 1237–46
29. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Book (2004)
30. Di Nitto, E., Dubois, D.J., Mirandola, R.: Self-aggregation algorithms for autonomic systems. Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd (Dec. 2007) 120–128
31. Saffre, F., Tateson, R., Halloy, J., Shackleton, M., Deneubourg, J.L.: Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications. The Computer Journal (2008)
32. Nakano, T., Suda, T.: Applying biological principles to designs of network services. Appl. Soft Comput. **7**(3) (2007) 870–878
33. Prenhofer, C., Bettstetter, C.: Self-organization in communication networks:principles and design paradigms. IEEE communication magazine (2005)
34. Di Nitto, E., Dubois, D.J., Mirandola, R.: Overlay self-organization for traffic reduction in multi-broker publish-subscribe systems. In: submitted for publication. (2009)
35. Babaoglu, Ö., Meling, H., Montresor, A.: Anthill: A framework for the development of agent-based peer-to-peer systems. In: ICDCS. (2002) 15–22
36. Cugola, G., Nitto, E.D., Fuggetta, A.: The jedi event-based infrastructure and its application to the development of the opss wfms. IEEE Transactions on Software Engineering **27**(9) (2001) 827–850