Robustness and Dependability of Self-Organizing Systems - A Safety Engineering Perspective

Giovanna Di Marzo Serugendo

Birkbeck College, University of London
dimarzo@dcs.bbk.ac.uk

Abstract. This paper analyses the robustness of self-organizing (engineered) systems to perturbations (faults or environmental changes). It considers that a self-organizing system is embedded into an environment, the main active building blocks are agents, one or more self-organizing mechanisms regulate the interaction among agents, and agents manipulate artifacts, i.e. passive entities maintained by the environment. Perturbations then need to be identified at the level of these four design elements. This paper discusses the boundaries of normal and abnormal behaviour in self-organizing systems and provides guidelines for designers to determine which perturbation in which part of the system leads to a failure.

1 Introduction

Self-organizing artificial (engineered) systems are appealing because they provide a "natural" robustness to changes and failures, while being composed of relatively simple entities. This claim, although backed by observation through simulations or experiments, has not been thoroughfully investigaged. For instance, questions such as: To what changes in their environment/faults are these systems naturally robust to, and what changes/faults are they not able to overcome (naturally)? What does "naturally" mean in this context? The boundary between the normal behaviour and the abnormal one is usually blurred since the self-* part of these systems contains (built-in or intrinsic) recovery capabilities. So, what is the normal operational mode of such systems, what is their abnormal one? This is also pointed out by Alderson et al. [1], in the context of complex systems: "Robustness is the invariance of [a property] of [a system] to [a set of perturbations]". The main point here is that a given system preserves a specific property for a specific set of perturbations, but may be fragile for *another property* or *other perturbations*.

The aim of this paper is twofold. First, it intends to clarify the notions of "normal", "dependable" and "resilient" behaviour in self-organizing systems. Second, it guides the designer of self-organizing systems in identifying the limits of the "natural" robustness, i.e. in identifying the *properties* and set of *perturbations* that render the system fragile. This is similar to the Failure Mode and Effects Analysis (FMEA)[1] technique followed by safety engineers when they analyse the

[1] http://en.wikipedia.org/wiki/Failure_mode_and_effects_analysis

design of a system to find what faults can occur. Starting from a block diagram of the system, the safety engineer considers what happens when each block fails and subsequently proposes changes to the system to make it safer. In the context of self-organizing systems, the designer determines which changes/faults in which part of the system lead to a failure. To this end, the paper lists the *design elements* of a self-organizing system, *properties* of self-organizing systems, *types of faults* for self-organizing systems and considers *means to reach dependability* in self-organizing systems. To illustrate our discussion, we consider a simple self-organizing systems and discuss its dependability to different changes and faults. This paper does not intend to be complete, additional investigations, experiments and measurements are necessary. The main goal of this paper is to lay the foundations for additional thorough investigation into these systems' behaviour.

2 Dependability and Resilience

This section provides a short summary of Avizienis et al. [2] and Laprie [3] papers. We extract (almost verbatim) here the essential elements of these papers that suit the purpose of our discussion and render our paper self-contained as we will refer to these notions in the next sections.

Robustness and Dependability. A computing system is *robust* if it retains its ability to deliver a service in conditions which are beyond its normal domain of operation [4]. *Dependability* is the ability to deliver a service that can justifiably be trusted.

Dependability Attributes. Dependability is measured against the following criteria: *Availability* - readiness for correct service; *Reliability* - continuity of correct service; *Safety* - absence of catastrophic consequences on the user(s) and the environment; *Integrity* - absence of improper system alterations; *Maintainability* - ability to undergo modifications and repairs.

Threats to dependability. A *failure* is an event that occurs when the delivered service deviates from correct service (service does not comply with functional specification). It is a transition from correct service to incorrect service. A service failure means that one or more of external service states deviates from the correct service state. The deviation is called an error. An *error* is a part of the total state of the service that may lead the system to its subsequent service failure. A *fault* is the cause of an error.

Means to attain dependability. *Fault Prevention* encompasses the improvement of development processes in order to reduce the number of faults introduced in the produced systems. *Fault Tolerance* aims at failure avoidance and is carried out through error detection and system recovery. *Fault Removal* occurs during system development or during system use. *Fault Forecasting* consists in evaluating the system's behaviour with respect to fault occurrence.

Resilience. In a recent paper, Laprie [3] provides an insight into the notion of resilience and its relationship with dependability. The considered systems are ubiquitous systems and the main point is to maintain dependability in spite of continuous changes. *Resilience* is then defined as the persistence of dependability when facing changes.

Self-organizing systems permanently face changes; this definition of resilience thus applies directly to self-organizing systems.

3 Self-Organizing Systems

Self-organizing applications are applications generally made of *multiple autonomous entities* with a knowledge limited to their *local environment* and that *locally interact* (directly or indirectly) to produce a result. Autonomous entities usually work in a *decentralised manner*, the global behaviour (function) generally "emerges" from the local interactions of the different entities. The entire "global" function is encoded in none of the individual entities. The result is generally obtained when the system reaches, converges to, a stable state. Typical examples of natural self-organizing system include swarms (ant, flocks of birds, wasps, etc.), immune system, human social behaviour (markets, trust, gossip). Artificial (engineered) systems include unmanned vehicles, swarms of robots, P2P systems, immune computer or trust-based access control.

3.1 Design Elements

Our discussion starts with the following consideration driven by design concerns [5,6]. The elements of a self-organizing (SO) system are: the *environment* in which it evolves (operating system, physical world, or network), the autonomous individual active entities - the *agents* - that constitute the system itself (software agent, robots, peer nodes), the *self-organizing mechanism* defining the rules (all) the agents apply (continuously) when evolving in the environment and letting them (re-)organise in case of changes or failures, and the *artifacts*, which are the passive entities maintained by the environment, created, modified and/or sensed by the agents (e.g. digital pheromone spread in the environment or information exchanged among agents).

$$\boxed{\text{SO System} = \text{Environment} + \text{Agents} + \text{SO Mechanism (rules)} + \text{Artifacts}}$$

Agents evolve into an environment, which they use to interact and carry on their behaviour. The boundary between an agent and its environment must be identified, but may vary from system to system. Depending on the system, it may be more convenient to consider an agent as a piece of software and everything else as its environment (in particular the underlying operating system or the node the agent is residing in). In other cases, it is more convenient to consider the agent as the combination of the piece of software *and* the node it is executing in. This is the case, when the agent is an autonomous (maybe mobile) robot, or when the agent is a node itself. Artificial systems usually take inspiration from nature - the SO mechanism being an ad hoc translation of the natural SO mechanism.

Sections 5 provides an example of SO Systems (stigmergy) identifying the environment, the agents, the SO mechanism, the artifacts together with examples of respective failures. Babaoglu et al. [7] describe this mechanism as well as others under the form of patterns and analyse in details these mechanisms and their corresponding implemented algorithms. Additional patterns for self-organisatin can be found in [8].

3.2 Types of Faults in SO Systems

In order to identify faults arising in a SO system, it is then convenient to consider faults (individually or as a combination) arising from each of these elements, i.e. faults from the environment, from the agents, from the SO mechanism itself, or from the artifacts, as shown in Figure 1.

Environmental Faults include all network related faults and communication faults arising among the agents; operational faults of the computing entities (nodes) present in the environment; any storage related fault (database, memory problem) affecting agent programs or artifacts; as well as any faults from the physical world in which the agents evolve (hole in the ground, unexpected obstacle). Environmental faults are a type of Interaction faults, more precisely they are System Boundaries faults (External faults), those that "originate outside the system boundary and propagate errors into the system by interaction or interference".

Agents Faults cover development faults affecting agent code and behaviour; when the node, in which the agent program resides, is considered part of the agent, then node faults are also Agents faults. We distinguish those faults from the ones directly affecting sensing and acting capabilities of the agents (vision camera fault or robot arm fault). Finally, an agent can be maliciously faulty. Agents faults may be Physical faults (hardware or software fault) and/or Development faults introduce during the system development. Agents interact with

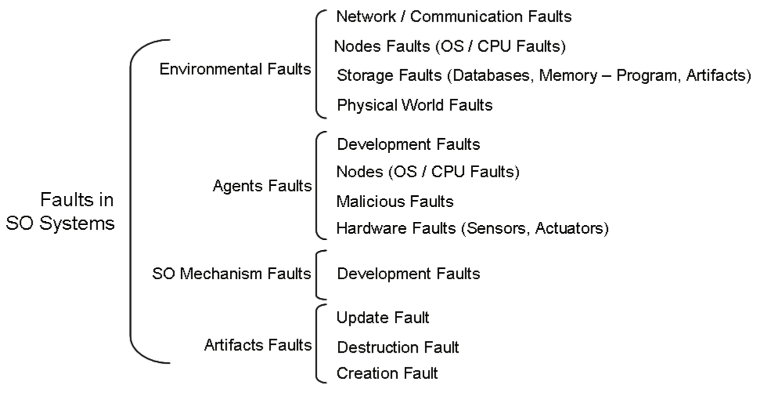


Fig. 1. Characterisation of Types of Faults in SO Systems

each other directly or indirectly through the environment. Agents faults may then also contribute to Interaction faults. Both Environmental and Agents faults may be the result of some malicious intent.

SO Mechanism Faults are Development faults, essentially due to errors in the design and implementation of the SO mechanism rules.

Artifacts Faults are all those faults affecting the integrity of the artifacts. Artifacts faults are likely to be caused by the Environment, the Agents or the SO Mechanism, since Artifacts themselves are essentially passive. We consider a fault to be an Artifact fault when it affects an artifact - through uncorrect modification, destruction, production or management of the artifact. For instance, an artificial pheromone whose evaporation rate is different than the expected one or that suddenly dissapears is an Artifact fault, in this case mostly caused by the Environment.

Environment and Agent faults may be either permanent or transient, while SO Mechanisms faults are permanent.

3.3 SO Systems Properties

We identify here the properties of self-organizing systems (Figure 2) that have to be questioned for the types of faults identified above.

Invariants. An invariant is any property that must be satisfied by the system at all time, i.e. it must be true at any state of the system.

SO Systems Robustness Attributes. *Convergence.* An important property of SO systems is whether the system actually converges towards the intended goal (correct value). *Speed of convergence.* How quickly does the system reaches its goal? *Stability.* Once the goal is reached, does the system maintain it? *Scalability.* How is the system affected by the number of agents and artifacts?

Dependability Attributes. These are the dependability attributes listed in Section 2: *Availability*, *Reliability*, *Safety*, *Integrity*, *Maintainability.*

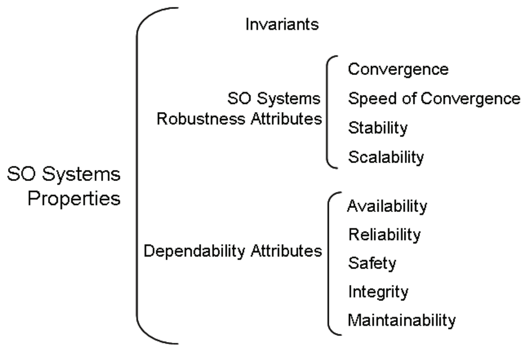


Fig. 2. SO Systems Properties

4 Normal vs. Self-Organizing vs. Resilient

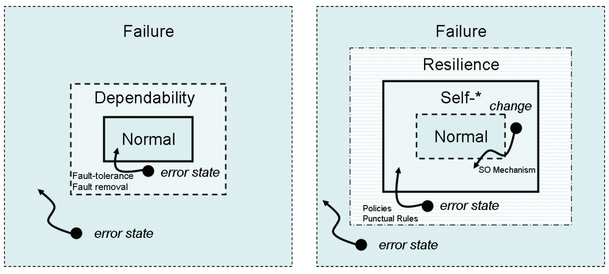
4.1 Dependability at Run-Time - Traditional Systems

Figure 3 (a) sketches the normal vs abnormal states in "traditional" systems (where no SO mechanism is involved). As discussed in Section 2, the means to reach dependability at run-time are: fault tolerance, fault removal and fault forecasting. In each case, this consists in *identifying the error state* and undertaking appropriate steps so that the system *goes back to a normal state*. The *Normal* operational mode of these systems occurs when the system is not in an error state. An identified error state triggers appropriate techniques (exception handlers, patches, etc.) to bring the system back to normal. *Dependability* thus extends to include all those error states. When the system cannot be brought back to normal, then the error state leads to a *Failure*, where the fault causing the error cannot be recovered.

4.2 Resilience at Run-Time - Self-Organizing Systems

The main difference between "traditional" systems and SO systems resides in the fact that an SO system recovers from an error *without error detection*, i.e. without specifically identifying an erroneous state and applying a specific recovery action.

Figure 3 (b) shows the different states of a SO system. *Normal* represents the ideal mode of operation of the system. The one when none of the faults discussed in Section 3.2 occur. *Self-** includes all the changes that the system overcomes without changing its mode of operations. These changes occur from the faults identified above. The SO system does not identify these changes as errors, it just carries on with its normal behaviour (that is why we do not call them errors). This is the area where the SO mechanism is enough to overcome the perturbation. In fact the *Normal* states could extend to the *Self-** ones. A SO mechanism has its limits, i.e. there are cases where carrying on as usual doesn't solve the problem (e.g. system does not converge or is unstable). *Resilience* thus refers to all the states where the SO system actually identifies an error and



(a) Traditional Systems (b) SO Systems

Fig. 3. Systems' States

specifically and *punctually* applies a recovery action. The system goes back to the *Self-** area again, where the usual SO mechanism rules apply. Finally, *Failure* refers to all the cases where the error cannot be recovered.

4.3 Analysis of Robustness and Dependability

In order to analyse robustness and dependability of SO systems, we need to determine which *perturbations* (change or fault) lead to which *property* being violated.
Properties. We consider the following properties:

```
Properties = Invariants - Robustness Attributes - Dependability Attributes
```

Invariants are all the invariant properties that the SO system has to preserve during its execution. *Robustness attributes* are convergence, speed of convergence, stability, and scalability (discussed in Section 3.3). *Dependability attributes* are availability, reliability, safety, integrity, maintainability (discussed in Section 2).

Perturbations. As we have seen above, the design elements of a SO system are the *Environment*, *Agents*, *SO mechanism* and *Artifacts.* The perturbations are any faults / changes / threats the system is likely to undergo originating from these elements (discussed in Section 3.2).

```
Perturbation = Changes or Faults in:
                    Environment - Agents - SO Mechanism - Artifacts
```

Analysis. Similarly to the FMEA technique, the designer needs to *question each element of the design* (environment, agents, SO mechanism and artifact), establish for each of them any *potential fault or change* that can actually occur, and *determine its impact on the properties* listed above.

```
Analysis = for all Design Elements
              for all Changes and Types of Faults
                 if change / fault can happen
                    is any Invariant modified?
                    is any Robustness Attribute affected?
                    is any Dependability Attribute affected?
```

An example of such a questioning could be:

> ”Is it possible that in the considered system the environment behaves maliciously, if yes then:
>
> – how is this affecting any invariant property (e.g. the value of a sum that has to be computed),
> – how is this affecting any robustness property (e.g. will the system still converge and at which speed), and

– what is the impact on any dependability attribute (e.g. is the service provided by the system always available or will there be disruptions)?"

Different faults, originating from different elements of the design, may have similar effects. For instance, a fault compromising the integrity of an artifact may be due to a fault in the environment (responsible to maintain the artifact), or to a malicious agent or to the artifact itself. In order to correct the fault, it becomes important to determine the origin of the fault by identifiyng the appropriate design element responsible for the fault.

Once a change or fault and its effect is identifed, a mean to attain dependability has to be identified through design change or additional resilience mechanism (see below).

4.4 Means to Attain Dependability in SO Systems

Let us discuss here the four means to attain dependability, as identified by Avizienis et al. and reported in Section 2, for the specific case of self-organizing systems (Figure 4).

Fault prevention. The design of self-* algorithms can be verified, to some extent, with mathematical analysis, but *simulations* are the most preferred tool at the moment.

Fault tolerance. For "traditional" software, fault tolerance consists in detecting an error and subsequently recovering from that error (with a bunch of diverse techniques). As said above, this is where SO systems differ from "traditional" software. We distinguish two levels.

First, the intrinsic fault-tolerance: the *SO mechanism* is robust enough to recover from errors *without* explicitly detecting an error and subsequently recovering from it. The SO system then "naturally" recovers from states which are not part of the ideal mode of operation. If a source of food suddenly disappears in an ant-based system, the SO system just carries on exploring the environment for food until the system finds another source. The disappearance of the food

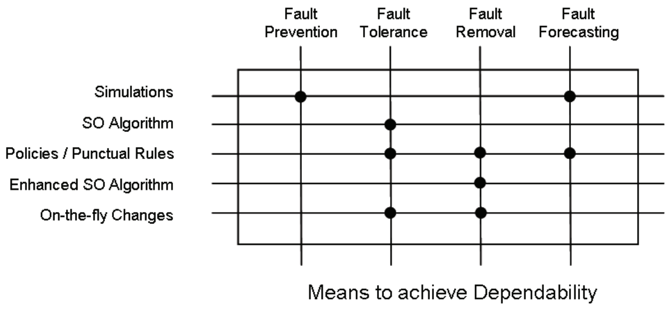


Fig. 4. Means To Attain Dependability in SO Systems

does not trigger any specific error, the system continues to run as in the normal case and just recovers (*Self-** in Figure 3).

Seond, similarly to fault tolerance in traditional software, the *Resilience* case expresses the limits of the SO mechanism to recover from errors. This is similar to "traditional" software: errors are detected (locally or globally) and appropriate measures apply in order to tackle the error. *Policies or punctual rules* then apply in order to recover from the error. These rules are different from the rules of the SO mechanism that are applied permanently by the entities of the system.

Fault removal. This encompasses: *enhanced SO mechanisms* with policies becoming part of the rules, or modified rules (so that what was before an error is now part of the normal operational behaviour); *on-the-fly changes*, such as switching among different SO mechanisms (rules) or adapting the rules, depending on the environmental conditions, or replacing outdated SO mechanism or policies with new ones on the fly during system execution.

As an example of enhanced SO mechanisms, we can mention the case of optimisation problems. The original Particle Swarm Optimisation algorithm detects one static optimum only, but is not able to cope with multiple or dynamic optimums. QSO is an algorithm that overcomes this problem, but still there is the problem of the swarm getting stuck in one optimum [9]. Multi-swarm is a solution to this that allows to find all optimums [10].

Fault Forecasting. This is similar to the *Resilience* case, through monitoring, exceeded thresholds are identified and policies punctually recover before the system reaches an error state.

5 Stigmergy - Ant-Based System

This example is a simulation of an ant colony foraging (see Figure 5 (a)).

SO System Elements. The *environment* is the physical world where ants evolve, where their nest is positioned, and where food is available. Ants deposit pheromone in the environment for marking paths food. The nest diffuses also a scent which helps the ants go back home with pieces of food. The *agents* are the ants. The *SO mechanism* works as follows:

– Ants are either looking for food, or going back to the nest once they have found food.
– When looking for food, ants leave the nest and walk randomly until they sense a pheromone scent in their locality. They then move in the direction where the pheromone scent is stronger.
– When they have found food, ants go back to the nest following the nest's scent. They follow the nest's scent in the direction where it is stronger.
– When they go back to the nest with food, they drop a pheromone scent at each step. This pheromone scent adds up to any other pheromone scent already present at the same place.

Finally, the *artifacts* are: the nest (in the center), the three food sources (upper-left corner, bottom-left corner and middle-right), pheromone scent (marking the path from food to nest) and nest's scent. The pheromone has an evaporation rate (how long it lasts) and a diffusion rate (how far it can be sensed). The pheromone is updated regularly by the environment (diffused and evaporated).

As we see from this description, the environment plays an important role when the SO system employs a SO mechanism using indirect communication such as stigmergy. Agents rely strongly on the environment and an Environmental faults can lead to a failure. In this example, the environment must host the pheromone and update it properly.

This system has no particular invariant, we list here the robustness attributes.

SO Systems Robustness Attributes. They are as follows:

- Convergence takes two dimensions here.
  Exploration: ants explore their environment properly - entirely and regularly - so as to spot any source of food.
  Exploitation: ants eventually bring back all the food to the nest.
- Speed of convergence:
  Exploration: how quickly ants can spot a new source of food once the current one is exhausted; *Exploitation*: how efficiently they can get the whole source back to the nest.
- Stability: ants focus on exploiting a source of food.
- Scalability: convergence and speed of convergence are not affected by the number of ants

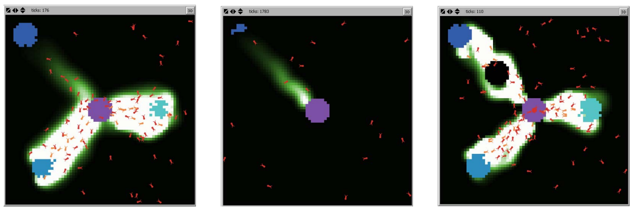
We started from the original simulation of [11], which is part of the NetLogo package. We altered it so as to insert different types of faults. We report here on some experiments we made in order to illustrate our discussion. A thorough investigation of the algorithm would require more experiments and precise measurements.

Real world applications taking advantage of stigmergy include static and dynamic optimisation problems as well as coordination of unmanned vehicles [12].

5.1 Environmental Faults

Ants disappear (or die). The system continues finding food, it converges but at a slower pace. If the number of agents is very low, then the pheromone path is not maintained and exploitation is less efficient, stability is compromised, but all food is eventually retrieved (Figure 5 (b)). This is similar to an agent crash and can also be seen as an Agent Fault.

Obstacle (Physical World). An obstacle (hole or rock) is now part of the environment mid-way between the upper-left source of food and the nest. The nest's scent on the obstacle is very low. Agents lay down the pheromone around the obstacle, thus adapting the path to find the food (Figure 5 (c)). None of the properties seems to be affected by this fault.



(a) Normal (b) Ants Die (c) Obstacle

Fig. 5. Normal Behaviour and Environmental Faults

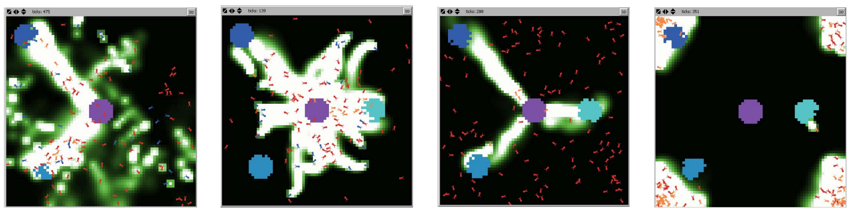
5.2 Agents Faults

Malicious agents. A subset of agents (25%) are not looking for food, but deposit pheromone at the wrong place (25% or 100% of the time) Depending on the quantity of wrongly added pheromone, paths to the food are more or less compromised (speed of convergence is slower and lack of focus on a source of food compromises stability). The system eventually converges and exploits all food. (Figure 6 (a) and (b)).

No Pheromone. Agents look for food, bring it to the nest, but do not deposit pheromone at all. Agents just look for food at random. All food is eventually retrieved but slowly. This is similar to the evaporation rate of the pheromone that is too quick.

5.3 SO Mechanism Faults

Pheromone Scent. Agents take a wrong direction when detecting the pheromone scent. As a result, agents avoid the paths leading to the food. Paths tend to



(a) Malicious Agents (b) Malicious Agents (c) Wrong direction - Pheromone (d) Wrong direction - Nest

Fig. 6. Agents and SO Mechanism Faults

disappear. There is no systematic exploitation. Food is eventually brought back to nest, but the system converges slowly (Figure 6 (c)).

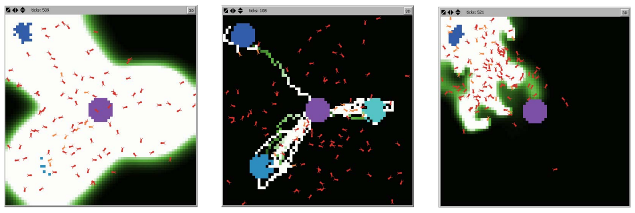
Nest's Scent. Agents take a wrong direction when detecting the nest's scent. Agents avoid the nest, do not find it, cannot deposit food and remain stuck at the opposite of the nest. The system does not converge at all (Figure 6 (d)).

5.4 Artifacts Faults

Evaporation rate of pheromone. Rate of 0% (or too slow): the pheromone scent does not evaporate (or not quickly enough), it stays where it has been laid down. The environment gets filled with pheromone, the ants continue following the paths even when food is exhausted. The system converges (it eventually retrieves all the food), but exploitation is not efficient (Figure 7 (a)). A small evaporation rate (above 6%) is enough for maintaining the paths without filling the environment with unnecessary scent. Rate of 100% (or too quick). Pheromone evaporates before ants can build a path and maintain it. Similarly to above, the system converges but is not efficient (speed is slow and stability is compromised).

Diffusion rate of pheromone. Rate of 0% (or too thin): the paths are thin and do not build up fully. A small rate (10%) is enough to construct solid paths (Figure 7 (b)). Rate of 100% (or too large): paths are large, ants do not go straight to food.

Nest's scent. The environment disperses the nest scent. In the simulation we first put random values instead of increasing values leading to the nest. Ants do not find the nest quickly anymore. Pheromone scent starts filling the whole environment. Efficient exploitation is compromised, but ants eventually exhaust all the food. Second, the nest's scent is randomised in a restricted portion of the environment, between the upper-left corner source of food and the nest, that portion of the environment is filled with pheromone (Figure 7 (c)). Efficient exploitation is compromised, but ants eventually exhaust all the food.



(a) Evaporation Rate - Slow

(b) Diffusion Rate - Thin

(c) Nest Scent

Fig. 7. Artifacts Faults

Food disappearance. This is a case of a change in the environment instead of a fault. After a short while the pheromone leading to the disappeared source of food vanishes, the ants just continue looking for food as usual, find and exploit other sources of food.

5.5 Analysis of Resilience

This example has no invariants, we discuss here robustness attributes; dependability attributes are discusssed in the next section. There are basically 3 categories of perturbations that affect robustness attributes: those that affect speed of convergence and stability (ants are moving randomly or not focusing on a source of food, thus taking longer to exhaust it); those that do not particularly affect the system (paths are maintained); and those that compromise convergence. Globally, we can say that when the system converges with a convenient speed of convergence and stability behaviour, it behaves normally (*Normal* box of Figure 3(b)). When the system eventually converges (despite slow speed of convergence and/or instability), the SO mechanism succeeded in overcoming the perturbation (*Self-** box of Figure 3(b)). If the system does not converge, an invariant is not preserved, the speed of convergence is too slow to be acceptable or if the instability becomes an issue, then we reach the limits of the "natural" robustness. Extra resilience is needed to support the SO mechanism. For instance, in the case of the ants, they may detect that they do not follow a path but go at random, and may decide to lay down another type of pheromone. This could overcome malicious ants trying to confuse them with pheromone deposited at the wrong place. Regarding scalability, a low number of ants affects speed of convergence and stability, while a large number helps building paths to find food. A large number of sources of food scattered all over the environment may also affect speed of convergence and stability.

6 Discussion

From the examples we have investigated so far, we can draw the following preliminary conclusions regarding the robustness and dependability attributes.

Invariants and Robustness. Convergence and invariants are key elements to determine the dependability limits of an SO system. A system that converges and maintains its invariants despite perturbations "naturally" overcomes those perturbations. An SO system needs additional resilience techniques when convergence cannot be reached, invariants are not satisfied, or speed of convergence and stability are not acceptable.

Availability. SO systems are always ready to work, but the service they provide may not be correct at first. It may take a certain time before the system converges.

Reliability. SO systems usually imply latency. Therefore, reliability is not necessarily ensured: a service is (necessarily) discontinued while the SO system re-organises/adapts to the new conditions. It may not stop, but will not be correct.

Safety. SO systems usually overcome a large range of changes/faults. However, the adaptation may imply latency. During this period, safety may not be guaranteed. In addition, in some cases the SO system is at a loss of overcoming the problem and may get stuck in a bad situation.

Integrity. Artifacts are at a high risk of integrity concerns/issues. They are not necessarily equipped with specific protection and are vulnerable because agents need the environment to exchange them, to modify or maintain them. Agents themselves are also at risk of integrity: they depend on the environment for their execution, their data or their physical movements.

Maintainability. We have seen that SO systems naturally adapt to changing software / hardware on-the-fly. In the case of SO system, we can contemplate changes at each level: changes in the environment, the agents, the SO mechanism and the artifacts in order to determine the maintainability level of the system.

7 Conclusion

This paper discusses the notions of robustness and dependability in the context of self-organizing systems. It proposes to analyse robustness and dependability by identifying perturbations (changes or faults) arising from each design element and studying their impact on invariants, robustness and dependability properties. Many research issues related to dependability of SO systems need to be investigated. Among others testing and formal verification of SO systems, or fault removal on the fly, which has not received much attention yet (e.g. switching SO mechanism, adapting the rules, applying specific policies to SO systems).

Acknowledgements

The author thanks John Fitzgerald for fruitful discussions on dependability and resilience as well as the anonymous reviewers for their feedback and comments on the paper.

References

1. Alderson, D.L., Doyle, J.C.: Can complexity science support the engineering of critical network infrastructures? In: IEEE International Conference on Systems, Man and Cybernetics, SCM 2007 (2007)
2. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
3. Laprie, J.C.: From dependability to resilience. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008 - Fast Abstracts) (2008)

4. Anderson, T. (ed.): Resilient Computing Systems. Collins (1985)
5. Ricci, A., Viroli, M., Omicini, A.: Programming MAS with artifacts. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2005. LNCS (LNAI), vol. 3862, pp. 206–221. Springer, Heidelberg (2006)
6. Picard, G., Gleizes, M.P.: The ADELFE Methodology-Designing Adaptive Cooperative Multi-Agent Systems. In: Bergenti, F., Gleizes, M.P., Zambonelli, F. (eds.) Methodologies and Software Engineering for Agent Systems, pp. 157–175. Springer, Heidelberg (2004)
7. Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G., Ducatelle, F., Gambardella, L., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology for distributed computing. ACM Transactions on Autonomous and Adaptive Systems 1(1), 26–66 (2006)
8. De Wolf, T., Holvoet, T.: Design patterns for decentralised coordination in self-organising emergent systems. In: Brueckner, S.A., Hassas, S., Jelasity, M., Yamins, D. (eds.) ESOA 2006. LNCS (LNAI), vol. 4335, pp. 28–49. Springer, Heidelberg (2007)
9. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization - an overview. Swarm Intelligence 1, 33–57 (2007)
10. Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Transactions on Evolutionary Computation 10(4), 459–472 (2006)
11. Wilensky, U.: NetLogo Ants model. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston (1997), `http://ccl.northwestern.edu/netlogo/models/Ants`
12. Sauter, J.A., Matthews, R., Parunak, H.V.D., Brueckner, S.A.: Performance of digital pheromones for swarming vehicle control. In: 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), The Netherlands, pp. 903–910. ACM, New York (2005)