Following the Problem Organisation: A Design Strategy for Engineering Emergence

Victor Noël and Franco Zambonelli

Abstract To support the development of self-organising systems, we explain and rationalise the following architectural strategy: directly mapping the solution decomposition on the problem organisation and only relying on the problem abstractions for the design. We illustrate this with an example from swarm robotics.

1 Introduction

Complex systems are made of simple elements and are characterised by the presence of non-linear interactions between them, no central control and the appearance of emergent behaviours at the system level [12]. In particular, emergence is the appearance of high-level behaviours resulting from low-level simpler rules [5] and an important mechanism governing these systems is self-organisation: autonomous change of the elements organisation without external control [5]. Multi-Agent Systems (MAS) is one field where self-organisation and emergence are studied and applied to engineer self-adaptive systems [6]. Some aspects of the global functionality are not explicitly pre-designed but emerge at runtime through this self-organising process in an endogenous and bottom-up way: the agents are unaware of the organisation as a whole [14]. Here, we assume self-organisation as the principle followed to design the low-level rules that lead to emergence.

In this paper, the general challenge is engineering self-adaptive self-organising complex systems that exist in and modify a complex context while meeting complex requirements, in the continuation of [1, 4].Towards that goal, we propose to look at practical methodological guidelines to accompany their design. In the following, we use the term "Self-Organising MAS" (SOMAS) to denote such engineered system.

The contribution of this paper is proposing and rationalising the following design strategy: when designing SOMAS, it is necessary to follow the problem organisa-

Victor Noël · Franco Zambonelli
DISMI, University of Modena and Reggio Emilia, Italy,
e-mail: {victor.noel,franco.zambonelli}@unimore.it

tion. It means mapping the SOMAS decomposition on the problem decomposition in elements (and not sub-problems), and relying only on the problem abstractions for the behaviours. This strategy is not a method by itself but a complement to existing approaches and methods. It is illustrated with a running example: a swarm of bots exploring and rescuing victims in an unknown environment. Everything presented is fully implemented (See `http://www.irit.fr/~Victor.Noel/unimore-ascens-idc-2014/`).

2 Following the Problem Organisation

2.1 Problems, Requirements and Design Constraints

A problem to answer is made of a context and requirements: engineering is finding a software solution, here the SOMAS, satisfying the requirements in that context [10].

As an example, in a robotics scenario, we look at the search and secure problem: bots must explore an unknown environment to look for victims and then secure them (supposedly to rescue them, but this is not covered in our example). The context is composed of the bots (controlled by the software to build) with limited communication capabilities, the environment that have walls, the victims that must each be secured by several bots. The requirements are to search and secure victims, to secure them all, as fast as possible, to explore the accessible space fairly, to completely explore the space in a non-random way, etc.

It is important to highlight the distinction between the problem answered by the choice of using self-organisation and emergence, and the design constraints it implies: existing works characterising self-organisation and emergence do not usually explicit this distinction [5, 6, 11, 14].

The following can be part of the problem: to have self-adaptation, a distributed deployment context, large-scale system, non-existence of an efficient centralised solution or impossibility of expressing the global behaviour of the system [3]. Inversely, the following are mandatory design constraints when engineering SOMAS: the fact that the decision must be distributed and decentralised, that the global macro-level behaviour and organisation can't be predefined or that self-organisation must be a bottom-up process initiated locally by the elements of the system [11].

2.2 The Strategy

When designing SOMAS, two main activities are of importance: decomposing the solution in agents and giving them a self-organising behaviour. It is usual in software engineering to first model the problem space before entering the solution space [10]. However, here, we closely map the solution decomposition in agents on the problem organisation, and only rely on the problem abstractions to design their behaviours.

By problem organisation, we mean the identification of the various elements participating in it and of the role they play with respect to the requirements. We call

it the "organisation" to avoid confusion with the meaning usually associated to a decomposition of the problem in sub-problems.

In the robotic example, the elements participating in the problem are the bots, but also the victims and the environment. Then, in relation with the requirements, the elements play the following role in the problem: bots moves to directions they choose, bots communicate with other bots, bots perceive victim, bots perceive walls, victims are situated, victims need a specific number of bots to be secured, etc. Inversely, an example of a potential decomposition in sub-problems would describe the problem as being about exploring and discovering on one hand, and securing collectively on the other hand.

The problem organisation can be imposed by the context (that the engineer can't control or modify) or must be chosen by the engineer when building the system. How to do so is an open question that we don't answer here.

Based on this modelling of the problem, we map elements of the solution (software agents) to the elements of the problem, and we give them the same capabilities as in the problem domain and not more. Their behaviour should be designed locally with respect to the relations elements have in the problem. The decisions (including those of their self-organising behaviour) they take should only rely on the problem domain abstractions and no higher-level global abstractions should be introduced.

In the robotic example, bots must choose where is the best direction to go at every given moment. For that, they can use what they directly see (victims and explorable areas), and when they don't know what to choose, they should rely on information shared by other bots about the state of the world with respect to the problem: where they are needed for victims or exploration. Hence, bots that see victims or explorable areas advertise about it. This information can be propagated by the bots and they can use it to decide where to go next.

Of course the complexity of the context and of the requirements (e.g., high number of bots, unknown scattering of the victims or limited perception means) are likely to make all these choices difficult. Correctly choosing the best action to take is thus an important questions: we don't pretend to answer it in this paper, but, as said before, we argue that such decisions must rely on the problem domain abstractions. Still, we comment on this question in the next section.

2.3 Relation to the Design of Self-Organisation

The strategy presented in the previous section can thus be used to design a SOMAS, but, as we highlighted it, is not enough by itself. In particular, a very important point is the problem of taking the correct local decision for the agents. Some approaches to self-organisation propose local criteria to be followed by the agents in order to drive the self-organisation. For example, the AMAS theory [9] is such an approach. Its main design strategy is that agents must have a cooperative social attitude: the whole approach rests on the theory that if the agents of a system are cooperative with the system environment as well as internally, then the system will behave adequately with respect to the objectives of the agents and of their environment. By identifying local non-cooperative situations agents can face, the engineer designs the agents so

that they prevent or correct such situations in order to put the system in a state as cooperative as possible. Usually, a measure called criticality that is shared amongst agents is used to reflect the importance of some state of the problem and to give an agent a way to decide between several choices.

In the robotic example, a bot often has the choice between several directions and do not see any victims. In order to take the most cooperative decision, he needs some information about the state of the system: bots can advertise for example about the direction they chose to go to and some measure (the criticality) of how much more bots are needed in this direction. When a bot propagates this information (because he chose the direction), it will update this criticality in order to reflect his and others participations in the self-organisation process: its choice means that this direction is a bit less critical now. Because bots assume they are all cooperative, they know that a direction chosen by another bot is presently the most important one to go to: choosing the most critical direction amongst all the neighbouring advertisements is enough for a bot to decide where to go next. Every decision taken will then influence how the bot computes this criticality, and inversely.

The way the self-organising process can be designed with this approach heavily relies on the fact that the agents does not contain any pre-defined behaviour in relation to the expected global behaviour, but only concepts manipulated in the definition of the problem itself, which serves to base the local decision on. For example, the criticality measure used in the AMAS approach reflects some aspect of the problem state in a comparable form: no extra high-level characterisation of what is or not a good global solution is used.

2.4 Rationale

The rationale behind the defended strategy, namely to follow the problem organisation when designing a SOMAS, relies on the design constraints highlighted in Section 2.1 and can be discussed in two cases: why follow the problem for decomposition and why use the problem organisation as we defined it here.

If the the engineer of a SOMAS introduces extra concepts foreign to the problem organisation, this means that when facing local decisions, the agents must translate their interpretation of the current state of the problem to the extra abstractions. Going far from the problem implies that we pre-set how situations are interpreted by the agents: it prevents them from interpreting correctly unforeseen situations because the concepts they manipulate can't capture them. In other words, the farther the design is from the problem, the lesser adaptive the system will be, and the lesser adequate behaviours can emerge.

In the robotic example, if the bots are designed so that to explore, they move in the direction of a repulsion vector from other bots (a typical algorithm for bots dispersion), then the problem solved is not about exploring while securing anymore, it is about dispersing bots in an environment: for example, in a hallway, a stopped bot securing a victim will prevent other bots to go behind him. Inversely, if bots behave as explained before, when the collective would profit from dispersing, then bots disperse as a result of going in directions advertised by others where the less

bots are going and when there is only one direction to go (e.g., a hallway), bots just go there because it is the only advertised direction.

Then, a problem decomposition in sub-problems calls for solving each sub-problem separately (if it is not the case, then the decomposition in sub-problems is useless for the design and this is out of the scope of the discussion here). This means that the sub-solutions must then be integrated together in the agents, and such integration is embedding the complexity of the problem.

In the robotic example, if the bots have a behaviour to explore and discover victims, and another to go help other bots secure their discovered victims, it becomes very difficult to handle at the agent level the choice between going in a direction or not: it could be needed to secure a victim, but there may be already other bots going there, so it must gather information about that, and then it could not be needed because other bots are going, but then maybe there is more to explore behind the victim, so it should go anyway, except in the case where there is still enough bots going there for the same reason as it is, and so on...

This puts back the complexity of solving the problem at the agent level instead of making it the result of the collective behaviour: it is the very reason why the paradigm shift proposed by self-organisation and emergence engineering was proposed in the first place. This matter has been discussed many times in the literature (the "complexity bottleneck" [11]): we don't bring new arguments for it.

3 Related Works and Discussion

The question of engineering emergence has been studied in various contexts, we discuss some of them and show how they are different from our contribution. On the whole, there is three ways of engineering emergences: ad-hoc, reusing or following a well-defined methodological approached.

First, ad-hoc means there is no explicit rationale behind decisions taken during the design: this is clearly out of scope of the current discussion as we are interested in ways to improve the engineering of SOMAS.

Then, reusing is usually done through the reuse of existing self-organising mechanisms that are well-known and understood. The main example of that is nature-inspired self-organisation [8]. These works rely on approaches or mechanisms dependent to a certain class of problems: they are easier to apply and to reuse when possible, but in exchange it is needed to translate the concepts manipulated in the problem to the abstractions of the solution reused. It is on that point that it diverges from our work: this means that part of the original problem is lost during that translation, and we precisely advocate for relying extensively on the problem.

Finally, methodological approaches are the closest to our work in terms of motivation. We cited in Section 2.3 the AMAS approaches and similar strategies are for example well discussed in [7]. All these works mainly proposes way to design the self-organising behaviour of the agents but mostly don't discuss (or rationalise) the decomposition in agents of SOMAS: this is what we do here. Other works note that simulation can be used to accompany the engineering of emergence in order to

iteratively change the design with respect to the observed results: it is called "co-development" [2] or "disciplined exploration" [13]. Our contribution is well coherent with these approaches, but of a different nature, and show that it is possible to exploit the problem organisation to reduce the development effort of SOMAS.

4 Conclusion

In this paper, we defend the idea that the specificities of self-organisation and emergence rationalise the need for decomposing the system by following the problem organisation. Of course, such an absolute assertion must be instantiated differently depending on the actual problem to solve: at the very least, this strategy and rationale improve the understanding of the relation between the problem and the solution decomposition. There exist many more other issues to explore on this subject, like how to well model the problem organisation, and other related subjects, like how the problem and the solution decomposition impacts the decentralised decision making.

Acknowledgment

This work is supported by the ASCENS project (EU FP7-FET, Contract No. 257414).

References

1. Abeywickrama, D.B., Bicocchi, N., Zambonelli, F.: SOTA: Towards a general model for self-adaptive systems. In: WETICE Conference, pp. 48–53. IEEE (2012)
2. Andrews, P., Stepney, S., Winfield, A.: Simulation as an experimental design process for emergent systems. In: EmergeNET4 Workshop: Engineering Emergence (2010)
3. Arcangeli, J.P., Noël, V., Migeon, F.: Software Architectures and Multiagent Systems. In: M. Oussalah (ed.) Software Architectures, vol. 2, pp. 171–208. Wiley (2014)
4. Cabri, G., Puviani, M., Zambonelli, F.: Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In: International Conference on Collaboration Technologies and Systems, pp. 508–515. IEEE (2011)
5. De Wolf, T., Holvoet, T.: Emergence versus self-organisation: different concepts but promising when combined. In: S.A. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, R. Nagpal (eds.) Engineering Self-Organising Systems, *LNCS*, vol. 3464, pp. 1–15. Springer (2005)
6. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-organisation and emergence in mas: An overview. Informatica **30**, 45–54 (2006)
7. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A. (eds.): Self-Organising Software. Natural Computing. Springer (2011)
8. Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F.: Engineering self-organising systems: nature-inspired approaches to software engineering. LNCS (2004)
9. Georgé, J.P., Gleizes, M.P., Camps, V.: Cooperation. In: Di Marzo Serugendo et al. [7], pp. 193–226
10. Hall, J., Rapanotti, L., Jackson, M.: Problem-oriented software engineering: Solving the package router control problem. Transactions on Software Engineering **34**(2), 226–241 (2008)
11. Heylighen, F., Gershenson, C.: The meaning of self-organization in computing. IEEE Intelligent Systems, Section Trends & Controversies **18**(4), 72–75 (2003)
12. Mitchell, M.: Complexity: A guided tour. Oxford University Press (2009)
13. Paunovski, O., Eleftherakis, G., Cowling, T.: Disciplined exploration of emergence using multi-agent simulation framework. Computing and Informatics **28**(3), 369–391 (2009)
14. Picard, G., Hübner, J.F., Boissier, O., Gleizes, M.P.: Reorganisation and Self-organisation in Multi-Agent Systems. In: International Workshop on Organizational Modeling, pp. 66–80 (2009)