The Knowledge Engineering Review, Vol. 33, e16, 1–24.   © Cambridge University Press, 2018
doi:10.1017/S0269888918000176

Towards adaptive multi-robot systems: self-organization and self-adaptation

CHRISTOPHER-EYK HRABIA, MARCO LÜTZENBERGER and SAHIN ALBAYRAK

DAI-Labor, Faculty of Electrical Engineering and Computer Science, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany;
e-mail: christopher-eyk.hrabia@dai-labor.de, marco.luetzenberger@dai-labor.de, Sahin.Albayrak@dai-labor.de

Abstract

The development of complex systems ensembles that operate in uncertain environments is a major challenge. The reason for this is that system designers are not able to fully specify the system during specification and development and before it is being deployed. Natural swarm systems enjoy similar characteristics, yet, being self-adaptive and being able to self-organize, these systems show beneficial emergent behaviour. Similar concepts can be extremely helpful for artificial systems, especially when it comes to multi-robot scenarios, which require such solution in order to be applicable to highly uncertain real world application. In this article, we present a comprehensive overview over state-of-the-art solutions in emergent systems, self-organization, self-adaptation, and robotics. We discuss these approaches in the light of a framework for multi-robot systems and identify similarities, differences missing links and open gaps that have to be addressed in order to make this framework possible.

1 Introduction

Software systems and combined hardware and software systems are becoming more and more complex, even more if they are directly interacting with the real world. Visions and evolving paradigms like the *Internet of Things*, *Precision Farming*, *Smart Grids*, *Industry 4.0*, or *Smart Cities* are high-level concepts, which are based on large, distributed, and loosely coupled system architectures that are executed in dynamic environments. Practical examples for such applications are autonomous vehicle fleets that—one day or another—will coin the picture of our transportation network or the application of mobile robots in our everyday life.

Due to this uncertainty arising from dynamic environments with limited availability of information and complex interactions between the system entities, it becomes difficult for designers and engineers to anticipate all conditions, interactions, and side-effects a system will have to deal with, when the system is specified and developed. As a consequence, the systems requirements are incompletely expressed (Bonjean *et al.*, 2014), which in turn creates a demand for systems, especially in the field of multi-robot applications, that put emphasis on robustness and adaptivity rather than on behaving in an optimal fashion. Being able to adapt during runtime to uncertainty with unforeseen changes in the environment, altered requirements or malicious system components themselves allows dealing with this inevitable under-specification (Noël & Zambonelli, 2015). Furthermore, it is possible that such large system ensembles are exhibiting unforeseen, complex behaviour that cannot be observed in single system entities. Such behaviour is commonly referred to as emergent behaviour. It can be either beneficial or malicious, hence control, monitoring, and validation is indispensably required in order to keep the system stable.

The aim of this article is to identify ways, mechanisms, and approaches that can help to gear emergent behaviour of mobile multi-robot systems towards their designated purpose and thus to profit from a goal-oriented autonomous system that features a high level of robustness and adaptability.

To approach this aim, we survey existing work and compare approaches from fields that are dealing with the above-mentioned challenges. In particular we analyze concepts and approaches from the areas of self-adaptation, self-organization, emergent systems, and robotics. We especially broaden our view on non-robotic system because robotics shares common goals with the multi-agent software community (Kaminka, 2012), which features various contributions in self-adaptation and self-organization. In doing so, we aim to identify similarities, differences, missing links, and open gaps. We put particular emphasis on identifying ways to derive low level behaviour from the global system's perspective and have a closer look on how bottom-up and top-down approaches are related to each other. Finally, we put everything into the relation of a multi-robot system application and the integration with higher-level decision-making and planning. However, the presented analysis and the drawn conclusion are also valid beyond pure robotic systems, which we are focused on.

The remainder of the article is structured as follows: in Section 2, we elaborate on concepts that we briefly and informally introduce, namely: emergence, self-adaptation, self-organization, and swarm robotics. In doing so, we identify and detail on characteristics and challenges that can be found in each of the presented concepts. In Section 3, we distinguish our own survey from existing ones that were done in related or similar fields, in order to emphasize the requirement of this work. In Section 4, we have a closer look into general engineering processes and methodologies and in Section 5 we present practical realizations that were done by means of existing middleware frameworks that support the development of any kind of adaptive or self-organizing system. Subsequently, in Section 6, we take a detailed look into mechanism design and implementation. The analysis in above listed sections lists common strategies and does further consider if the solution is domain specific or independent, if it is a centralized or decentralized approach, if the approach has a specific application focus, like improving certain quality measures, or if it is developed with a multi-purpose perspective. Moreover, it is determined if an approach allows for the development of solutions from a top-down or bottom-up perspective. In Section 7, we forge the link to our particular application domain, namely robotics. In doing so, we focus on how identified mechanisms can be integrated into domain-specific planning and decision-making concepts. In Section 8, we highlight common characteristics, open challenges, and point out future research directions, before, in Section 9, we conclude our work.

2 Concepts

In this section, we briefly explain frequently used terms and their mutual dependencies to and relationships with the targeted problem domain.

2.1 *Emergence*

A beneficial utilization of arising global behaviours from distributed entities can be found in nature, where they are rather common. As an example, consider the collective intelligence of social insects, like ants and termites, which is used to find shortest paths, foraging, or to build temperature-balancing hives (Deneubourg *et al.*, 1990; Bonabeau *et al.*, 1999). Other higher developed animals are as well showing similar kind of collective intelligence, a popular example are fish schools, which increase the protection from predators for the individual (Viscido *et al.*, 2004). Such collective intelligence does not rely on individuals that control or co-ordinate the group.

This phenomenon is commonly referred to as *emergence*, with global (macro level) behaviour, patterns and properties arising from the interactions between local parts of the system (micro level). This is commonly constraint in the way that the macro-level structure needs to be novel. This implies that on micro level, there is no knowledge about the global or macro-level goal nor there are intentions to control the global behaviour of the entire system. Similar definition can be found in the literature, see Ali *et al.* (1998), Goldstein (1999), Camazine *et al.* (2003), De Wolf and Holvoet (2005).

The research about emergence has a long history and has diverse scientific roots, for instance, in cybernetics, solid state or condensed matter physics, evolutionary biology, artificial intelligence, and artificial life. Its aim is to develop tools, methodologies, and solutions that allow to understand and to reproduce processes which lead to emergence (Serugendo *et al.*, 2006).

De Wolf and Holvoet (2005) classifies the research into four schools of research, namely complex adaptive systems theory (Kauffman, 1995), nonlinear dynamical systems theory (Newman, 1996), synergetics (Haken, 1984), and far-from-equilibrium thermodynamics (Nicolis, 1989). A compiled history as well as analysis of different definitions can also be found in Serugendo *et al.* (2006).

The fascination of emergence derives from nature of the concept itself. Emergence can be characterized as *simple*, *robust*, and *adaptive*. Emergence is simple, because the individual entity or agent needs only a straightforward behaviour algorithm in order to play its part in a global system. The latter can be exceptionally complex. Emergence is robust, since the solution does not depend on a central single point of failure and agents can usually be replaced, removed, or added without changing the global behaviour. Finally, emergence is adaptive due to the several independent entities that can flexibly adapt and thus easily master changes that occur in an uncertain environment. However, the agent's behaviour algorithm is static and not supposed to be adapted.

2.2 *Swarm robotics*

In Section 1, we argued that simplicity, robustness, and adaptability are characteristics that are particularly interesting for the domain of (mobile) multi-robot systems—after all, multi-robot systems are supposed to operate in extremely dynamic and highly uncertain environments. Practical examples for challenging application areas are, for example, precision farming, traffic surveillance, nature conservation, and disaster rescue operations. All of these environments have one thing in common, that is: multi-robot systems have to achieve their goals autonomously, in a timely manner. In doing so, robot systems have to adapt to ever-changing conditions. The concept of emergence can significantly contribute to more effective multi-robot systems, especially in terms of scalability, robustness, and flexibility.

Furthermore, a system of several probably simpler robots can be easier adjusted to varying problem complexities, for example, by simply adding or removing agents. Multi-robot systems, which implement or adapt the concept of emergent behaviour, are commonly referred to as *swarm robotic systems* (Şahin, 2005). Another definition from Dorigo *et al.* (2004) is Swarm robotics consists in the application of swarm intelligence to the control of robotic swarms, emphasizing decentralization of the control, limited communication abilities among robots, use of local information, emergence of global behaviour, and robustness. Commonly, swarm robotic systems are classified as multi-robot systems consisting of simple robot individuals. A disaster rescue system with several aerial and ground robots capable to execute complex tasks individually is usually considered a multi-robot systems, whereas several simple robots with limited individual capabilities, for example, the robots are only able to move around and execute one primitive task, are understand as swarm robotic system. Nevertheless, the transition in between swarm robotics and multi-robot systems is fluent and not always consistent in the literature.

The concept of a 'swarm' is significantly based on the idea of homogeneity (Şahin, 2005). Nevertheless, concepts of 'heterogeneous swarms' can be found as well. Existing examples commonly employ different specialized entities (Ducatelle *et al.*, 2010; Dorigo *et al.*, 2013).

Swarm engineering is a subdiscipline of swarm robotics that deals with 'the design of predictable, controllable swarms with well-defined global goals and provable minimal conditions' (Kazadi, 2000). Swarm engineering was first formally defined by Winfield *et al.*, (2004) as a combination of swarm intelligence and dependable systems. In particular, this includes procedures for modelling, designing, realizing, verifying, validating, operating, and maintaining swarm robotics systems (Brambilla *et al.*, 2013).

The particular domain of controlling robot swarms has gained much attention over the last years. One rather popular sub-branch of this domain is human–robot swarm interaction (Naghsh *et al.*, 2008). *Human–robot swarm interaction* is commonly understood as a direct steering of, or influencing the swarm directly, in order to control its motions. This can be done by controlling special leader or predator entities directly, either being virtual or real, which are influencing the remaining swarm members

(Bashyal & Venayagamoorthy, 2008; Goodrich *et al.*, 2011). One refers to 'assistive swarming' (Penders *et al.*, 2011), whenever the human being is integrated into the swarm as the steering leader or predator. Although these approaches are beneficial for a direct interaction during a mission, they are not directly supporting the application's goals during design and development. Furthermore, these approaches mostly rely on behaviour rules that are specified at design time in a bottom-up approach, see, for example, Krupke *et al.* (2015).

Kloetzer and Belta (2006) argue that, in contrast to single robot system, the behaviour of swarm robots has to be specified more qualitatively. Following Kloetzer and Belta (2006), a swarm is naturally described by features like shape, size, and position of the region that is occupied, while the exact position or trajectory of each robot is not important.

2.3 Self-organization and self-adaptation

Very close to biologically inspired concepts of emergence and swarm intelligence is the idea of self-organization. The terms emergence and self-organization are frequently confused. The history of the term as well as the distinction between both concepts was comprehensively discussed by De Wolf and Holvoet (2005), who define the concept of self-organization as follows:

Self-organization is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control (De Wolf & Holvoet, 2005: 7).

Nevertheless, the term self-organization is also used as the process that leads to the state of emergence (Goldstein, 1999; Noël & Zambonelli, 2015).

Additionally, Serugendo *et al.* (2005) distinguish between strong self-organizing systems that do not have central explicit external or internal control, and weak self-organizing systems that might have some internal central control instance.

Considering self-organising and emergent systems as distinct concept they still have one thing in common, that is: there is no explicit external control whatsoever. Although the external influence on self-organized systems is more explicitly researched in the domain of *guided self-organization* (Prokopenko, 2009). In this context the external influence is distinguished in specific and non-specific, referring to a specific influence for a direct control on the spatial, temporal, or functional structure and to non-specific influence if the system still decides on its own how it reacts upon the external stimulus. In consequence Prokopenko (2009) defines the guidance of self-organization as possible limitation of the scope or extent of the structures/functions, or specification of the rate of internal dynamics, or selection of a subset of possible options that the dynamics may take. Furthermore, Ay *et al.* (2012) proposes external rewards, problem-specific error functions and assumptions about the symmetrics of the desired behaviour as possible strategies for guided self-organization.

Following De Wolf and Holvoet (2005) the main difference between self-organization and emergence is that in the case of the former, individual entities can be aware of the system's intended global behaviour. In consequence self-organization can be seen as a weak form of emergence. The intuitive and regularly used approach to realize self-organization is applying the concept of feedback loops. Here, parts of the system are monitoring the state, analyzing it in reference to the intended behaviour and triggering appropriate responses. This approach is also used for 'single entity systems'. In this case the concept is referred to as *self-adaptation* (Brun *et al.*, 2009; de Lemos *et al.*, 2013). However, if a decentralized system containing several entities exhibits adaptive behaviour to external changes this is as well considered self-adaptation. Self-adaptation in relation to software in general is defined as follows.

Self-adaptive software modifies its own behaviour in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation. (Oreizy *et al.*, 1999: 55).

2.4 Common characteristics

While the particular characteristics of emergence, swarms, self-adaptation, and self-organization may differ, the general objective of all of these concepts is exceptionally coherent, that is, having ensembles of robust systems that maintain their structure and feature a high level of adaptation. Here, structure can either be understood in the organizational sense in case of multiple systems or as the integrity of a single agent.

A system, which enjoys these characteristics, for example, appropriate adaptation capabilities that remain within predefined bounds (also referred to as 'controlled autonomy'), would lead to a simplified design, because not all possible states and state transitions have to be specified in advance. In fact, these ideas are in compliance with the visions of the promoters of autonomic (Kephart & Chess, 2003) and organic computing (Schmeck, 2005), respectively. Both concepts consider systems that promote self-adaptation as major property, facilitating the development of software that can manage itself at runtime. Furthermore, self-adaptation is considered to be the foundation of other envisioned self-properties, such as self-configuration, self-optimization, self-healing, and self-protection (Kephart & Chess, 2003).

Continuing this thought, it would no longer be necessary to exactly specify the low-level system behaviour in all possible situations that might occur, but rather, leaving the system with a certain degree of freedom to allow for autonomous reaction and adaptation to new situations in an intelligent way.

Still not answered are the questions, how such systems can be developed from a goal-driven perspective, especially how this can be integrated into more sophisticated agents that exhibit a high level of autonomy with advanced decision-making and planning capabilities.

In the following sections, we present and analyze available works that aim to answer the above-listed questions. In doing so, we discuss limitations of these approaches but also elaborate on their advantages and disadvantages. Our goal is to identify problems, opportunities, and challenges that have to be addressed in the future.

3 Related surveys

The field of emergence, self-organization and self-adaptation has already been surveyed from different perspectives. In the following we summarize the existing literature. In the course of this, we put emphasis on the limitations of these surveys in order to substantiate the need of this article.

The survey from de Lemos *et al.* (2013) presents state-of-the-art methods for engineering self-adaptive systems. de Lemos *et al.* (2013) distinguish between design space, processes, centralized and decentralized control, and practical runtime verification and validation. de Lemos *et al.* (2013) understand *design space* as which developers need to decide during the development of a self-adaptive system and see the main challenge of creating a systematic understanding of the options for self-adaptive control during the design process. The concept of *Processes* refers to influences of self-adaptive systems to the software system life cycle. Following de Lemos *et al.* (2013), activities are not bound to the traditional development-time, but are shifted to runtime. In the area of control architectures different possibilities with advantages and disadvantages are presented. It is emphasized that adaptation control is always realized with feedback loops, which comprises: monitor analyze plan execute (MAPE) in different combinations. Planning is albeit only considered regarding the control of the adaptation process and not in respect to the goal-oriented behaviour of the system.

The formulated question in this context is under what circumstances and for what systems the different patterns or architectures of control are applicable. For verification and validation, the open challenge is to design and deploy certifiable methods that are able to deal with the explosion of the state-space. Since they are focusing on very general aspects of engineering, their perspective is more on the abstract process of developing self-adaptive systems and less on how such systems can be realized in practice, for example, in the context of multi-robot systems.

Another survey paper focuses on the importance of feedback-loops in self-adaptive systems (Brun *et al.*, 2009). Brun *et al.* (2009) present several existing approaches from different fields like control theory, nature, and software—like the autonomic computing initiative. A generic feedback loop is proposed as well. This feedback loop acts as a refinement of the sense-plan-act mechanism and highlights that software engineering needs to develop an own understanding of feedback loops. Moreover, they introduce some specific challenges regarding the concepts of modelling, maintenance, middleware support and verification, and validation. In particular, Brun *et al.* (2009) substantiate the need to create reference libraries and architectures of control-loop types and mechanisms of control-loop interactions as well as to create appropriate middleware solutions with possibilities of validation and verification. It is also underlined that the impact on software maintenance of self-adaptive systems and the detection of malicious or unintended behaviour has not yet been addressed sufficiently. We generally agree with this argumentation and see similar requirements for self-organizing multi-robot systems, for example, providing middleware support for fostering code reuse. We discuss existing frameworks and approaches in Sections 5 and 7.

The survey from Serugendo *et al.* (2006) is about self-organization and emergence and the differences and relations of these concepts and the arising challenges. The paper is focused on mechanisms and definitions rather than on particular engineering techniques or frameworks. Focusing on multi-agent systems the authors classified existing mechanisms of realizing self-organization into five classes:

- Using direct interactions between agents using basic principles such as broadcast and localization;
- using indirect interactions between agents using the environment (stigmergy);
- using reinforcement of agent behaviours;
- using co-operation behaviour of individual agents;
- using a generic architecture.

Furthermore, Serugendo *et al.* (2006) formulate the central question, how individual agents can be programmed to exhibit self-organized behaviour as a whole. For this reason their highlighted major challenge is developing means to define global goals, and to design local behaviours so that the global behaviour emerges. The field of swarm engineering is surveyed by (Brambilla *et al.*, 2013). The authors review on the one hand side methods for designing and analysis and on the other hand collective behaviours that have been studied using swarm robots. It is argumented that the most studied fields are design and realization as well as verification and validation, which are still not sufficiently solved. Moreover, Brambilla *et al.* (2013) claim that other topics, such as requirements analysis, maintenance, and performance measurement have not yet gained much attention. The authors are also coming to the conclusion that there is still no formal or precise method to design individual level behaviours that produce the desired collective behaviour. The existing automatic design methods are further classified into the categories of evolutionary robotics and multi-robot reinforcement learning. However, the authors do not link to the problem of developing systems that are pursuing their goals and exhibiting self-organized behaviour at the same time.

In the work of Ye *et al.* (2017) a specific focus lies on self-organization mechanisms used in multi-agent systems. Specifically, the authors analyze existing literature from a objective-based perspective and focus on the research issues task/resource allocation, relation adaptation, organizational design, reinforcement learning, enhancing software quality, and collective decision-making. The authors explicitly elaborate about specific self-organization mechanisms that are used in above mentioned research issues rather than on general approaches, methodologies, or tools that foster the realization of self-organized systems. However, Ye *et al.* (2017) are also arguing that most research is theoretical and real applications are still in an early stage.

This paper continues the foundation of the existing surveys. In contrast to the recited surveys, this paper is starting from a common perspective, revisiting self-adaptation and self-organization architectures and methodologies in the different related fields. It also focuses on existing frameworks and middleware frameworks, and details on how the actual adaptive self-organizing mechanism is going to be realized in practice and how it could be integrated into the decision-making and planning capabilities of state-of-the-art robotic systems.

4 Methodologies

In addition to the presented results of the survey from de Lemos *et al.* (2013) about software engineering and self-adaptation in general, other authors have developed different concepts.

The software engineering process PosoMAS is introduced and compared to other existing agent-oriented software engineering methodologies in Steghöfer *et al.* (2014). The process practices are embedded into the risk-value life cycle of the Open Unified Process (OpenUP). It is targeting open, self-organizing systems and contains different practices for the design of agent architecture, agent organization, agent interaction, system architecture. The core practices are goal-driven requirements determination, pattern-driven multi-agent system (MAS) design, using existing architectural, behavioural, and interaction patterns from reference architectures, an iterative evolutionary agent design, model-driven observer synthesis, trust-based interaction design, and agent organization design. The *organization design* as well as the lifecycle phase *design system dynamics* are the aspects where self-organization and its consequences have to be considered by the user of the process. Unfortunately, there are no details about how this has to be done. The process only provides placeholders for the consideration of self-organization during system design.

In contrast to PosoMAS, more support is provided by the earlier and continuously developed agent-oriented methodology Atelier de Développement de Logiciels à Fonctionnalité Emergente (ADELFE) that provides a model-driven software development process for developing adaptive co-operative multi-agent systems (Picard & Gleizes, 2004; Bonjean *et al.*, 2014). The process is supported by a special notation AUML (Agent Unified Modelling Language), which is based on UML2 (Unified Modelling Language) as well as some tools and libraries. The process itself is separated into the stages of preliminary requirements, final requirements, analysis, design, implementation, and tests.

The authors propose that global behaviour is determined with testing and simulation but the methodology is not bound to this approach. ADELFE has still a very coarse view on the problem of developing self-adaptive and self-organizing systems, since the actual complexity of how to create the required behaviours, mechanisms, and algorithms is not addressed, even though the process provides more guidance than the PosoMAS approach.

Ideas from ADELFE are also integrated into the TROPOS4AS extension (Morandini *et al.*, 2009) of the agent-oriented software engineering methodology TROPOS (Bresciani *et al.*, 2004). TROPOS focuses on requirement analysis and models the system top-down in several analysis steps. The entire design process is supported with the TROPOS modelling language. Of particular interest are goals and how they are decomposed and delegated to the individual agents. TROPOS4AS extends TROPOS goal models to allow for the description of self-adaptive systems. This is achieved by additional annotations that allow to express environment specific conditions, goal creation and failure handling. TROPOS4AS incorporates ideas from ADELFE to model agent organizations with co-operation rules.

Gershenson (2007) presents a self-organization engineering methodology introducing the *mediator* as a central concept. The mediator is understood as an adaptation controller that is reducing the friction between parts of the system in order to fulfil the overall system goals. Reducing friction means increasing positive synergies and reducing inhibitions between the system parts. The mediator or adaptation controller is supposed to adjust the behaviours of parts leading to the global goal. The controller could be using methods like evolutionary algorithms or reinforcement learning. Furthermore, the author proposes a process consisting of the stages representation, modelling, simulation, application, and evaluation with iterative cycles between adjacent stages. The author states that this approach is neither top-down nor bottom-up. Based on the additional degrees of freedom of self-organizing systems, the author declares that they cannot be tightly controlled since they have own goals, instead they should be steered. In contrast to the above-presented approaches, the methodology of Gershenson (2007) is providing guidance for a suitable architecture of self-organizing systems. However, the support is still on a high level of abstraction, plus, it remains unclear how the user could determine or select appropriate mechanisms, mediators, or controllers.

Another high-level architecture is presented in the context of the Organic Computing initiative Branke *et al.* (2006). The proposed abstract observer–controller architecture is created for the realization of systems with self-properties. The observer collects data from the system and computes some indicators characterizing the global state and the dynamics of the system. It may use methods like classification in order to characterize the situation. The controller compares situation parameters with the goal, which was defined by the user, and decides whether an intervention is required and what action would be most appropriate. The controller is able to influence the local decision rules of the system agents, the system structure, the number of agents, or the environment. The controller interferes only when necessary and affects only some system parameters, it does not control single agents in detail. Decisions (mapping from situation to action) are based on learning. Learning can be facilitated by simulation. The presented architecture is actually only defining a centralized feedback loop for the system and abstracts from the intended behaviour, but how to define the behaviour is still an open issue. Moreover, the proposed learning approach requires the occurrence of errors. These can be problematic in cases where errors would affect the operability of a system.

In Noël and Zambonelli (2015) the authors propose an abstract approach based on component decomposition for engineering application-specific self-organizing systems. The authors argument that the identification of elements and the assignment to roles builds a design bridge between the problem and the emergent behaviour, because the selected decomposition limits possible macro-level behaviours that can be executed during runtime.

Contrary to the surveys we presented in Section 3, which have highlighted the importance of formal methods with validation and verification and contrary to the software engineering approaches that we discussed above, Edmonds (2005) argues that in order to create adaptive, self-organizing systems, experimental methods—as used in classic science—are required. The reason for this is that a traditional engineering approach supposedly cannot deal with unexpected changes during future runtime. In fact, the author argues that there is no general systematic or effective method that can generate or find a program to meet a given formal specification and determine whether it meets that specification. Instead Edmonds argues that systems should be accompanied with an open hypothesis database, which collects hypotheses that already have been tested—including all relevant test-conditions. Albeit, it is not mentioned how such a system could be realized in practical applications.

De Wolf and Holvoet (2007) focus more on the mechanisms themselves and describe the properties of some core self-organization mechanisms as design patterns and provide a guide when they can be applied and how they should be selected to reach a targeted goal. In particular they mention digital pheromones, gradient fields, market-based control, tags and tokens. However, only gradient fields and market-based control are discussed in more detail. Although, the paper is a helpful guide for developers of self-organizing systems, it leaves open how the actual implementation or class design could be realized.

In Fernandez-Marquez *et al.* (2012) the work of De Wolf and Holvoet (2007) is continued by analyzing, classifying and describing a set of bio-inspired self-organizing mechanisms. Here, the authors classified mechanisms and their relations into three layers of basic, composed and higher-level patterns. In that sense, the composed layer is created by a combination of basic mechanisms and the higher-level patterns show different options of exploiting the basic and composed mechanisms. On the basic level they identified the basic patterns of spreading, aggregation, evaporation, and repulsion that build the foundation for a realization of all composed and higher-level mechanisms. The created catalogue of patterns is intended to be used as a base for a more modular design and implementation of self-organizing systems.

The idea of reusable design patterns is also incorporated by Reina *et al.* (2014), who first introduced cognitive design patterns. However, cognitive design patterns are not focused on self-organization or self-adaptation. Instead, the concept considers mechanisms for collective cognition in distributed multi-agent system in general. In Reina *et al.* (2014) the authors present a in-depth study of a collective decision-making mechanism in order to use it as a cognitive design pattern independent of its application in future.

The review of existing methodologies shows that many approaches tackle the problem of designing self-adaptive or self-organizing systems on a very coarse process and architecture level and only identify the particular points where the designer would have to integrate related considerations. Albeit this is providing some support for system designers and developers, this seems not sufficient. On the other hand, Edmonds (2005), De Wolf and Holvoet (2007), and Fernandez-Marquez *et al.* (2012) are more specific regarding the actual challenges of guiding the development of such systems in practice. Edmonds (2005) supports a general paradigm shift that focuses on experimental methods instead of concentrating on provable correctness. Contrary, De Wolf and Holvoet (2007) and Fernandez-Marquez *et al.* (2012) provide reusable design patterns that can be applied to new applications. The support of an actual implementation in a more specific and application-oriented perspective is reviewed in the next section, where we discuss specific middleware solutions and frameworks.

5 Middleware solutions and frameworks

Several middleware solutions, frameworks, and infrastructures for the development of self-adaptive and self-organizing systems with different perspectives have been proposed. All have in mind to foster and support the development of adaptive systems that are able to deal with uncertainty either for single systems or multiple system entities. The specialities and similarities are discussed in the following.

Hernandez-Sosa *et al.* (2005) present an adaptation framework inside their component-based robotic framework with the focus on resources used by modules. Thus, their focus is purely adaptive quality of service (QoS) in order to keep the system robust and stable even in high load situations and not the general behaviour of the system under normal execution conditions.

Another self-adaptation framework with focus on resource usage is ReFrESH, which targets all kinds of embedded systems. This includes hardware and software (like robotic system) and is realized with an embedded virtual machine running on real time operating system (Cui *et al.*, 2014). Different to Hernandez-Sosa *et al.* (2005) the focus is especially on self-adaptation mechanism for fault detection and not QoS. The goal of the framework is an automatic reconfiguration of the system with other or additional components. The evaluation of the current state is tailored to the detection of abnormal resource usage. The adaptation does not include learning and is strongly focused on the described problem domain.

Rainbow is a more general architecture framework for self-adaptive systems (Garlan *et al.*, 2004) that is not limited to resource usage adaptation like former approaches. It focuses on single systems and does not consider multi-agent system environments. The core idea is the separation of the adaptation layer in order to foster reuse of adaptation mechanisms. Furthermore, their approach requires a definition of the adaptation mechanisms during design time based on preprogrammed adaptation operators and strategies. This seems feasible for very simple adaptations like parameter tuning, but does not satisfy the vision of systems adapting autonomously in complex versatile environments.

Differing from the resource focused frameworks and rather inspired by Rainbow, SodekoVS is a more general architecture framework, supporting software engineering of self-organizing systems (Sudeikat *et al.*, 2009). The SodekoVS library provides a catalogue of mechanism patterns as reusable components that provide systematic problem-oriented descriptions of mechanisms in an abstract, reusable format. This facilitates the selection and combination of mechanisms. Their multi-layer architecture consists of a top most application layer including standard application functionalities and may be linked to agent implementations for application-specific parts. The co-ordination layer placed below consists of the agents as well as a substrate, which can contain one or more co-ordination media. Mechanism instances are encapsulated in distinct co-ordination media and are interfaced by co-ordination components that allow to modify agent states. The presented architecture does not mention, how decision-making and planning could be combined with the self-organization mechanisms.

The application-independent description of (inter-)agent co-ordination patterns is supported by the domain-specific language MASDynamics that allows to map interrelations of agent activity to detailed agent design models (Sudeikat & Renz, 2009). However, the language describes the intended macroscopic behaviour only on a very abstract level, it is just defined by group count, roles and group membership count. The co-ordination pattern can be parameterized and configured based on agent events and internal state representations that can be passed through the co-ordination media. Nevertheless, it is still required to determine the actual mechanism manually. Furthermore, the authors propose a process for self-organization engineering with the stages of requirements, analysis, design, implementation, and test (using simulation), similar to the approach of Gershenson (2007) (see Section 4), but without explicit iterations. The focus of the framework lies on the engineering structure without considering the development of co-ordination mechanisms themselves nor the explicit integration with other higher-level decision-making and planning capabilities.

In analogy to the Rainbow and SodekoVS frameworks Preisler *et al.* (2013) developed a framework that separates the co-ordination of self-organizing MAS from the application logic. Similar to SodekoVS, the so called co-ordination space represents a separate and explicit layer in the architecture. Their goal was improving the reuse and exchange of co-ordination mechanisms. The co-ordination itself can be distributed over several nodes by using information forwarding with mechanisms like publish and subscribe. Further, the co-ordination is written in a external co-ordination model using also the domain-specific language MASDynamics (Sudeikat & Renz, 2009) with mechanism described in declarative fashion. This language allows for a more structured description of co-ordination using concepts of roles and links between agents. Nevertheless, the presented framework does not provide a mechanism for deducing or selecting the mechanism itself from given goals. Additionally, the authors are not linking the co-ordination components with the agent's decision-making and planning, like Sudeikat *et al.* (2009).

jSwarm is a middleware for cyber physical systems in general with an explicit focus on swarm robotics (Graff *et al.*, 2013, 2014). It allows for centralized sequential programming with spatial-temporal constraints with concentration on the motion in space of robot systems. The application code is analyzed, scheduled, and distributed according to a centrally computed dependency graph among the system entities. This enables the replacement of individual robot motion by an application migration between different robots for increasing the system performance, while considering spatial-temporal constraints. The used service-oriented architecture hides heterogeneity and diversity and abstracts from the particular resources. How the specific swarm behaviour based on the constraints for individual entities is designed is not further considered. In fact, jSwarm provides an abstract infrastructure for a centralized controlled distributed swarm robot system. Even though it fosters adaptability by reassigning robot applications to other robots, the focus is more on a performance optimization and less on enabling more robust and adaptable distributed systems.

The framework TOTA (Tuples on the Air) particularly focuses on the aspect of information propagation in a decentralized self-organizing system (Mamei *et al.*, 2005). Their approach from the field of sensor networks is inspired from biological morphogen gradients (cells react based on thresholds that change while propagating) and does not need global perception, distance, and direction sensing. TOTA provides abstractions and mechanism to support the creation of distributed overlay data structures spread across a mobile network. It is composed by a dynamic *ad hoc* wireless network of possibly mobile nodes, each capable of locally storing tuples of information and letting them diffuse through the network. The consideration on how influences propagate is important for large scale real-life applications and not considered by most of the other frameworks.

Another more application specific work comes from the field of modular robotics, describing robots consisting of several independent modules. The paper shows how distributed constraints can be used to build adaptive modular robots (Yu & Nagpal, 2009). Their framework abstracts from simple sensors and actuators and uses distributed constraints combined with neighbourhood sensing for distributed adaptation. Open questions are how the required constraints or control laws (sensor feedback functions) can be developed to achieve the entire system goal, how several goals are going to be addressed in parallel, and how these distributed constraints can be combined with the robots' decision-making and planning.

TuCSoN is a platform for the development of self-organizing systems (Viroli *et al.*, 2009). It is based on the linda tuple-space model and provides Java agents with tuple centres (tuple spaces enhanced with a reactive, rule-based programming model), which can be used to implement probabilistic and timed co-ordination rules in a declarative style. It is designed around the formulated requirements for self-organized co-ordination. In particular, *topology* describes that each agent is directly connected to a small set of neighbourhood locations. *Locality* defines the local interaction between agents and the co-ordination media and between two co-ordination media. The requirement *Online character* says that co-ordination is triggered by interaction or by elapsed time. *Time* ensures that co-ordination rules generally timed. The last requirement *probability* defines that co-ordination is always non-deterministic.

The rules in TuCSoN are represented with templates as identifiers for the matching mechanisms. In case of several matching templates the selection is randomized. The reactivity is realized with reaction tuples that are triggered through formulated conditions of source, target, status, time, and goals.

Altogether, TuCSoN has a much stronger focus on the co-ordination mechanisms themselves and allows for the implementation of data-centred self-organization co-ordination algorithms. Nevertheless, it does not include support for reaching the goal of the system developer. In consequence the developer still needs to figure out on its own, how to move, transform, or copy data in order to achieve a certain pattern. Moreover, the authors do not combine their approach with other application-specific decision-making and planning.

A subset of the described patterns in Fernandez-Marquez *et al.* (2012), see Section 4, is implemented in the execution model BIO-CORE (Fernandez-Marquez *et al.*, 2011), which provides basic bio-inspired services, namely the basic patterns evaporation, aggregation, and spreading as well as the gradient pattern. BIO-CORE consists of three main parts: a shared data space that allows to exchange data, basic bio-inspired services implementing basic bio-inspired mechanisms, and interfaces providing primitives for the agents to interact with the core.

Buzz is a domain-specific language that provides a middleware for the simplified implementation of swarm robot applications (Pinciroli & Beltrame, 2016). In Buzz a set of robots (swarm) is a first level object that can be used for group task assignment and set operations (intersection, union, difference, and negation). Furthermore, Buzz has capabilities for neighbourhood operations (queries, filtering, virtual stigmergy) and information sharing. The language runs on an own virtual machine. Nevertheless, the realized mixed bottom-up and top-down approach does only provide an environment for developers wherein self-organization and cognitive algorithms can be implemented. A disadvantage of Buzz is that developers need to leave their well-known ecosystem with tools and programming languages for learning a new language that is limited in expressiveness.

The work presented in this section shows various directions, the approaches are either very specific to a particular domain or problem, like resource sharing or information propagation, or are general frameworks that simplify the implementation of abstract concepts, like presented in Section 4. The important problem of implementing the actual mechanisms and algorithms in general is only addressed in TuCSoN (Viroli *et al.*, 2009; Fernandez-Marquez *et al.*, 2011). However, it is still open how a developer could solve a particular problem from a goal-oriented perspective, and how the co-ordination and adaptation part can be linked with the agents' decision-making and planning. In the next section we go into detail and analyze works that explicitly focus on the support of designing and implementing the required mechanisms for adaptive systems.

6 Mechanism design and implementation

We already highlighted that, besides the architecture and the engineering process, the mechanism itself plays a key role for self-organization or self-adaptation. The challenge of implementing these mechanisms is also raised in several above-mentioned references (Kephart & Chess, 2003; Schmeck, 2005; Serugendo *et al.*, 2006). Facilitating the development of these mechanisms is a crucial requirement for a suitable framework. Thus, in this section we focus on existing means of development.

The power and simplicity of co-ordination mechanisms is proven in nature and existing phenomenons are extensively studied in the field of swarm intelligence (Bonabeau *et al.*, 1999). Using similar approaches led to a number of promising results especially for swarm robotics. For instance, Masar (2013) created an algorithm for swarm exploration and surveillance based on a combination of a swarm particle algorithm, using the popular force field paradigm with ant colony algorithms. Other concepts are based on behaviour descriptions (Balch & Arkin, 1998), rigid virtual structures (Ren & Beard, 2004), and graphs (Desai *et al.*, 2001). In comparison to the numerous force field approaches, the latter methods are less flexible and are computationally more expensive.

Unfortunately all these approaches have the disadvantage that they are inherently problem and application specific and that they were created in bottom-up manner by means of trial and error testing. None of the approaches was inferred or selected from a goal description, respectively using top-down engineering.

We classified existing work that deals with design and implementation of self-organization and self-adaptation mechanisms into the categories of evolution, learning and simulation approaches, functional languages, declarative, and application-specific approaches. These classes are discussed in the following subsections and completed with a consideration of the mechanisms analysis, verification, and validation.

6.1 Evolution, learning, and simulation

Some attempts of determining mechanisms of self-adaptation and self-organization are in consensus with the biological inspiration and are based on the concepts of evolution and learning. They are either implemented in a simulation or directly shaping the agents' behaviour online during execution.

Das *et al.* (1995) have used genetic algorithms to automatically generate application-specific synchronization strategies. Even though the presented approach is still depending on a centralized timer for the calculation of updates in the distributed entities it defines a promising direction. The particular challenge is the formulation of an adequate utility function and evaluation environment for a general application of this approach.

Matarić (1995) describes an approach of controlling group behaviour of robots by synthesizing a particular complex behaviour based on a set of primitive behaviours. On that account unsupervised reinforcement learning was used on a basic behaviour set to hide low-level control details to steer the global group behaviour towards the global goal. The authors also shaped the reinforcement function by partitioning the goal into subgoals for providing more immediate feedback. In experiments the authors have been able to implement a foraging task based on the empirically derived set of basic behaviour of avoidance, following, aggregation, dispersion, homing, and wandering. Open questions are, how the initial behaviour set is going to be selected, how such a system would be initialized in order to avoid possible harmful state conditions in the early stages of the reinforcement learning process, how the goals are partitioned, and how this might be combined with individual tasks of the robots.

Evolutionary robotics (Nolfi & Floreano, 2000) is a special domain of robotics that especially researches how to automatically design robot controllers. The evolutionary idea is as well applied in swarm robotics (Kernbach *et al.*, 2008). Depending on the particular approach the researchers use algorithms from machine learning, like artificial neural networks (Dorigo *et al.*, 2004; Groß *et al.*, 2006), combined with evolutionary-programming, genetic programming (Groß *et al.*, 2006) combined with physics simulations, before generated controllers are deployed to the real robots.

Francesca *et al.* (2015) are working on a general purpose solution that generate the individual controllers of robots in a swarm, therefore the authors evaluated the performance of different approaches to design distributed control software for robot swarms in two empirical studies including several tasks. The evaluation of manual, manual with constraints and three automated control generation approaches is done using simple e-puck robots. The solution vanilla is using F-Race optimization, chocolate is using an iterated F-Race optimization, and for comparison EvoStick uses an artificial neural network approach without hidden layers that directly connects sensors and actors. The simulation environment ARGoS is applied to determine the fitness of particular configurations against a utility function. The AutoMoDe approaches vanilla and chocolate are generating a probabilistic finite state machine by combining and fine-tuning preexisting parametric modules. In Francesca *et al.* (2015) these approaches are compared against human designers, which are allowed to freely design the control algorithms as well as in another setting designing controllers constraint with the same limitations as the AutoMoDe approaches. The final results show that chocolate was able to outperform the human designers in the given controlled experimental environment. The presented result is promising, but it is still not clear if it can be generalized and applied to more complex scenarios. Especially the definition of a suitable utility function can be challenging task.

6.2 *Functional languages*

The approaches presented in this section have developed different functional languages in order to program the behaviour of distributed systems. Broadly stated, using formalized functional languages has the advantage that it is simpler to prove the correctness of the program.

An early attempt is presented by Klavins (2004) with a functional language CCL for distributed systems based on predicates. The language is formalized and could be generally proven for determining the correctness of a program, but this is not yet developed. Furthermore, there is no functionality that would allow to deduce single entity behaviour from a global mission description, requiring an individual implementation of each agent.

The functional language of Beal and Bachrach (2006) allows for programming sensor networks in a top-down fashion. Here the meta-code is compiled to bytecode that is going to be executed on a platform-specific virtual machine on the nodes. They are able to compile the global behaviour description, which is written in Proto language, into locally executed code that produces the intended emergent phenomena. The validation of the result is accomplished with simulation. Nevertheless, the focus seems more on aggregation features for abstracting communication between the nodes, the actual required rules still need to be developed in a manual way and bottom-up fashion.

The programming language Protoswarm is an extension of Proto. It allows to program robot swarms moving in space as single continuous spatial computer (Bachrach *et al.*, 2008). This approach provides an abstraction that enables global programming of a decentralized system. A disadvantage of this approach is, that it requires homogeneous swarms with all agents exhibiting the same behaviour. This makes it very difficult to combine these behaviours with other individual tasks of the agent. Furthermore, Protoswarm has no explicit support for verification and validation.

6.3 Application-specific approaches

A simplification of the problem domain can be achieved by concentrating on the application specific requirements. Existing approaches are discussed in the following paragraphs.

The origami-inspired programming language (Origami Shape Language) is able to translate its instructions into gradient rules that lead to self-assembly of the programmed geometric structure for a large set of identical agents (Nagpal, 2002). It relies on a set of simple primitives that are all related to gradient diffusion and neighbourhood gradient sensing. It is difficult to apply the folding language to real life problems, since it is hard to imagine how to fold the geometric structure in order to reach a desired shape. Furthermore, the approach also requires some kind of global knowledge, because each agent needs to know if it belongs to a certain area.

Similar to the previous paper, Joshi *et al.* (2014) are also addressing the specific application of self-assembly. They present a robotic system that allows to build a desired structure in a decentralized fashion with automatically generated rules for the single agents. The agents perform independent local sensing and co-ordinate their activity through stigmergy of the perceived structure. In their solution all robots have a representation of the intended global target and use qualitative stigmergy, meaning actions are triggered by qualitatively different stimuli, such as distinct arrangements of building material. In order to derive the local rules (movement guidelines) for the single robots an offline compilation step is recursively searching through the state space to identify structpaths without cycles that are meeting the requirements. This is possible, since the state space is reduced by using some global constraints, like all robots are starting from the same start point, a fixed movement direction and a fixed robot behaviour routine.

Another work focused on modular robots, consisting of several independent agents, shows a generalized distributed consensus framework that allows for collective self-adaptation by using simplified local constraints (Yu & Nagpal, 2011). The authors mathematically proofed the convergence to the intended goal state from all initial states and determined factors that are influencing the convergence speed, namely number of agents, task complexity, agent failure, and agent reactivity. The authors used their framework to implement different self-adaptive modular robots, for instance, a gripper and an automatically balancing bridge. In their discussion they stated several limitations and future research directions. In particular, their approach is only working for a single-consensus state, thus, fulfilling a goal state of two independent sensory information would not be possible. Furthermore, tasks are assigned in advance and dynamic task selection is not available. Another difficulty is the determination of the local constraints from the global objective function that could be addressed in future research by implementing a task compiling framework for automatic constraint generation. Finally, the authors propose that future research could benefit from a mixed strategy of decentralized and centralized control.

6.4 Declarative approaches

A different and more general direction is taken from other researchers investigating the applicability of declarative approaches. In this context logic programming is used to describe global goals and restrictions of the system and allow to deduce the rule set of individual agents.

Kloetzer and Belta (2006) developed a hierarchical framework for defining swarm motion behaviour. It is based on linear temporal logic, constraints, and predicates. The authors abstract a large continuous geometric state space that needs to be fulfilled by the swarm entities. The abstraction is achieved by describing control constraints with mean and variance. The linear temporal logic formulas are automatically mapped to provably correct robot control laws. A disadvantage of this approach is that it requires a global knowledge of the system state.

In the field of modular-robots there is a promising programming language called Meld (Ashley-Rollman *et al.*, 2007). In the fashion of logical programming languages, Meld uses a collection of facts and a set of production rules for combining existing facts. Execution facts are combined to satisfy given rules and to produce new facts that—in turn—can satisfy additional rules. This forward chaining process is executed until all provable facts were proven. However, the rules meeting the application goals have to be determined manually by the developer.

An additional perspective is given by another paper presenting the LDP (locally distributed predicates) language. LDP was designed to develop distributed modules in a robot ensemble, so called modular robot programming, from a global perspective (De Rosa *et al.*, 2008). A LDP program is a collection of LDP with associated actions that are triggered on any agent that matches the predicate. Their reactive programming approach does not consider even simple control structures and parallelization. As a consequence the realization of more complex tasks becomes very challenging. Furthermore, their approach has not been evaluated with regard to self-organization.

A property-driven design approach for swarm engineering is introduced by Brambilla *et al.* (2012). In this iterative top-down process the requirements of the system are defined with logic formulas in the language PRISM. Based on the formal description a macroscopic model is generated and verified with a model checker. Next, the verified model is used to guide the implementation of the system using a simulator. Finally, the system is tested on real robot hardware. While this approach can help to create and implement a model of a swarm with desired properties, it becomes problematic to create more complex models with robot-to-robot interaction, time, and spatial aspects that cannot be easily expressed with logic formulas.

Montagna *et al.* (2013) presents a domain specific solution addressing the field of pervasive services. The introduced framework for self-organizing and self-adaptive systems is based on live semantic annotations (LSA). LSA are actually predicates with local predicates for individual agents as well as for the environment. Predicates are applied in inspiration from chemical formulas, a specific input leads to certain output. The global behaviour is enacted by defining rules in the LSA-space that are called eco-laws. All LSA in a distributed space are updated based on the context and the eco-laws, which are defined and represented as the set of LSA stored in a given locality. They identified a set of basic rules to model mechanisms: *evolution, decay, diffusion, contextualization, composition, synthesis*. This set is not complete, but in their opinion is sufficient for supporting low-level patterns. Furthermore, the authors developed resource description framework (RDF) as language for structuring LSAs and coding eco-laws. RDF relies on the SPARQL/SPARUL query languages. The whole approach is tailored to service domain with a comparable less complex environment that is less prone to uncertainty and noise. Applying this solution for instance to the field of multi-robot systems would require to model the complex real-world environment with LSAs, which seems to be a very challenging task.

SCEL (Service Component Ensemble Language) is a formal language based on knowledge stored in tuple spaces (Nicola *et al.*, 2015). The interaction (behaviours) with the tuple space can be guarded with policies. The language itself is minimal and does not included higher level control constructs, but allows for semantic verification using model checking. The extensions Policed-SCEL adds direct support for the policy language FACPL and StocS enables SCEL semantics. Due to its minimal characteristic the expressiveness of SCEL and its extensions is limited, but decision-making can be integrated using external reasoners but this is not part of the formal language yet.

6.5 Analysis, verification, and validation

In connection with the question of how we can develop appropriate adaptation or self-organization mechanisms is the challenge of analyzing, verifying, and validating the created results. Due to the emergent characteristics and additional degrees of freedom it needs to be ensured that the solution does not exhibit undesired, malicious, or dangerous behaviour. This is particularly important to make adaptive systems widely acceptable for users (Schmeck, 2005).

The best option of generating respective guarantees is using mathematical proofs as they are achievable with logic programming and formal languages, like Meld (Ashley-Rollman *et al.*, 2007) and SCEL (De Nicola *et al.*, 2013), shown in Section 6.

In the discussed paper of DeRosa *et al.* (2008) the authors refer to logic programming languages for distributed systems like Meld. They argue that these tools are powerful, because programs written in them have certain provable properties, but this provability limits the expressive range of the languages. A similar perspective is described by Edmonds and Bryson (2004), who state that self-organization performance cannot be evaluated by purely formal methods. This argumentation is based on the assumption that reasoning alone cannot process all the information required to predict the behaviour of a complex system (see Section 4). Edmonds and Bryson (2004) argued that verification of code, as a pure design methodology, is only suitable for very simple problems and does not work for larger problems. This is proven with the halting problem of Turing. They propose that especially because of the large amount of time spend on debugging application, we should better spend time on validation through experimentation. This is called experimental methodology, like it is done in classical science like chemistry. In fact they recommend that modules like agents should be accompanied with a set of testable hypothesis and data sets which have been tested with the modules.

Even the authors of Meld argue that representing the state space can be problematic. Moreover, dealing with a continuous state space including probabilities and temporal constraints can be difficult. For example, expressing a behaviour that is able to react to perception of a fuzzy sensor, like a computer vision system, and expressing a response with a certain intensity or speed cannot easily be expressed with logic programming.

A different perspective is taken by De Wolf *et al.* (2005). They show an analysis of self-organizing emergent application behaviour based on numerical analysis. It is executed on discrete simulation steps instead of mathematical equations. This allows for more reliable and valuable results in comparison to just observing simulation results. A critical point is how to identify the important macroscopic properties and their related variables that quantify the property, because these properties are specific for each application. Furthermore, DeWolf *et al.* (2005) are referencing Wegner (1997), who showed that pure formal descriptions of the macroscopic behaviour of complex interacting systems are impossible in general.

To sum up, a pure formal verification of correctness is only partly applicable, since it relies on a complete description of the state space and hence is less useful for unknown uncertain conditions. Other approaches using simulations and experimental results are limited as well, but could possibly complement a formal verification.

7 Planning and decision-making in robotics

Self-organization is often applied in swarm robotic systems to model collective behaviour, such as collective decision-making and task allocation (Brambilla *et al.*, 2013). However, existing approaches entirely focus on the global perspective of the system without considering how this it integrated with the individual obligations of a robot, which might not be directly related to the global collective behaviour. In order to incorporate self-adaptation and self-organization into the individual behaviour control of mobile robots in order to benefit from the advantages of those concepts it is necessary to integrate and combine them with existing approaches for individual decision-making and planning. This is necessary since especially more sophisticated intelligent robots still have to pursue their mission of solving particular problems while they need further capabilities for self-adaptation and self-organization in dynamic environments. In the following we evaluate existing works regarding the applicability and support of dynamic environments.

As discussed in Hrabia *et al.* (2015), adaptability in general, and fast and flexible decision-making and planning in particular, are crucial capabilities for autonomous robots. The widely applied Robot Operating System (ROS) proposed by Quigley *et al.* (2009a) and the many existing third-party packages illustrate the current state-of-the-art in applied robotics research. In the ROS community the most commonly used approaches for task-level or behaviour control are rule-based and reactive. A popular package is *SMACH* that allows the user to build hierarchical and concurrent state machines (Bohren & Cousins, 2010). All kind of state machine based approaches have the problem that a decision or reaction can only be given if a state transition was already modelled in advance. The more task-oriented behaviour trees, available in the *pi_trees* package (Goebel, 2013), are another popular alternative. Even though behaviour trees allow for more reusable behaviour implementations and some higher-level abstractions, by using special operators like sequencer and selector, the path of execution is still preprogrammed and is likely to fail in dynamic environments. The reason is that it is difficult to design such system without missing possible useful dynamic transitions.

More flexible is the belief desire intention-based implementation *CogniTAO* (CogniTeam Ltd, 2015) available in the *decision-making* package that also includes modules for the creation of finite and hierarchical state machines. CogniTAO is targeting high-level control of multi-robot systems in dynamic environments. In CogniTAO the user defines behaviours, called plans, as decision-graphs with start and end conditions. Behaviours are executed and selected by protocols. The selection and decision process is synchronized through a team of robots. The concept is more suitable for uncertain environments because the execution sequence is not fixed and the selection of behaviours (plans) is based on conditions. Nevertheless, it is still difficult to define mission or maintenance goals and there exist only simple protocols for plan selection.

Outside of the ROS community we can find other reactive approaches, for instance, the subsumption architecture (Brooks, 1986) that is using a hierarchy of controllers. The controllers are generating output signals and are able to suppress or influence the signals of lower levels. Finally, signals are combined and mapped to particular actions. The system is able to deal with uncertain situations since it always creates reactions, but it is still hard-wired and neither can address dynamic goals nor sequential task dependencies.

Influenced by the concept of artificial neural networks is the behaviour network architecture (Maes, 1989). Here, the behaviours are connected with each other based on their preconditions and effects, which enables dynamic state transitions, receptively a stateless action model. The executed behaviour is dynamically selected based on a utility function. The utility function is calculating the positive influence on a goal for each behaviour, called activation that is spread through the network. Different extensions of this approach introduce learning capabilities and multi-robot application (Jung, 1998) or concurrency and non-binary preconditions (Allgeuer & Behnke, 2013). The hierarchical version of Allgeuer is available as a ROS package, but simplified in the way that it only relies on pre-computed static inhibitions of conflicting behaviours. A general issue is that network parameters need to be properly tuned towards a specific application to avoid the risk of getting stuck in cycles, since it is difficult to express a required execution order.

Symbolic planners in the tradition of STRIPS (Fikes & Nilsson, 1972), like hierarchical task networks (Breuer *et al.*, 2012) or further advanced implementations, like graph-based approaches (Blum & Furst, 1997) or heuristic planners (Bonet & Geffner, 2001; Hoffmann, 2001) are very good in directing towards a goal and determining a possible execution order. However, they have problems under uncertain conditions because of high computational costs and calculating alternative decisions in case of unreachable goals.

In order to combine the advantages of both, reactive behaviour approaches, being fast and more flexible, and classical planning, being more goal-directed, some researchers have proposed hybrid architectures. An early attempt was given by Hertzberg *et al.* (1998), who proposed a low level behaviour controller using two differential equations, one for creating the actuator control signal and one controlling the activation. The integration of a special term in the activation equation enabled external influences from an operator like a human or a classical planner. Here, the reactive behaviour layer is still operational on its own and the model for the planner has to be provided independently.

A more recent hybrid approach (Lee & Cho, 2014) models each high-level task as an independent behaviour network that is selected by a planner. This architecture is strongly goal-directed, but loses flexibility for dynamic adaptation since behaviours of different networks cannot be combined with each other.

In our opinion hybrid approaches that combine behaviour networks with symbolic planning are the most promising concept to create robots that are adaptive and goal-driven at the same time. The support of dynamic state transitions as well as relying on a reactive agent model provides a good foundation for the future integration of self-adaptation and self-organization capabilities in a multi-robot setting. The presented literature includes many fruitful ideas (Maes, 1989; Hertzberg *et al.*, 1998; Jung, 1998; Allgeuer &

Behnke, 2013; Lee & Cho, 2014), even though they are not yet integrated into one solution, and even though not all of them are available for the ROS community.

8 Opportunities and challenges

Despite the fact that there is some progress and interesting work done in the community, there are still many questions that remain open. Table 1 allows for an aggregated comparison of all approaches that were mentioned in this article. The comparison shows that existing approaches become increasingly specific (that is domain-specific and application-specific) the closer one looks into the actual mechanisms and algorithms that are required to exhibit adaptive and co-operative behaviour. Hence, we observe that—on a methodical level—all solutions are domain independent. Methodologies that can be classified as 'more specific' are very close to the evolutionary emerged natural role models of self-organizing systems that have been developed with trial and error, respectively, in a bottom-up manner, over centuries. For instance they apply learning, simulation, and experimentation or put existing natural algorithms into reusable patterns.

All mechanism designs and implemented solutions that do not apply mechanisms in an *ad hoc* and bottom-up manner are confined to a specific domain and sometimes even centralistic. The complete centralization is exceptionally cumbersome whenever the system scales up. This is mainly due to possible communication and computation bottlenecks as well as having a single point of failures that limits performance and robustness for larger systems.

Available middleware solutions and frameworks are also often limited by their rather centralized or abstract approach or focus on particular problems, like system performance or resource management. Furthermore, none of the more concrete approaches tries to address the problem of creating an adaptive, self-organizing system in a goal-oriented manner. A general purpose approach to develop the required mechanisms is still missing, even though some promising solutions for simple robot swarms with limited capabilities exists (Francesca *et al.*, 2015). We do not know yet, how such kind of system can be designed and developed from a goal-oriented perspective in a way that an intended global behaviour is deduced to individual agent behaviours. Furthermore, as yet, it is unclear how the required self-organization mechanisms can be automatically determined from the application's perspective. Even though declarative approaches such as Ashley-Rollman *et al.* (2007), Brambilla *et al.* (2012), Montagna *et al.* (2013), Nicola *et al.* (2015) allow for some deduction and validation of the mechanisms, they lack of expressiveness and cannot be used to model uncertain complex environments.

Existing architectures dictate that the engineer is implementing the actual agent behaviours from a bottom-up perspective or the approach is very limited to a particular use-case and limited in its expressiveness. A framework architecture that allows for a goal-driven approach using a top-down, or mixed bottom-up and top-down process is not yet available.

Moreover, when it comes to multi-robot systems, our target domain, existing solutions are either focused on a particular application (e.g. as robot self-assembly), thus, modular robots are centralized and inflexible or their capability is limited to encompass the conceptualization of the motion of a swarm system.

We were not able to find works that deal with the problem on how self-organization and self-adaptation can be combined with existing and indispensably required decision-making and planning of more sophisticated robotic systems that are not just focusing on the self-organization task itself.

Nevertheless, the analyzed literature shows a number of common ideas and concepts in order to further the spirit of more reusable code and simplified development of adaptive single- or multi-component systems. The most important ideas and concepts are:

1. The implementation of co-ordination and adaptation are separated from the application behaviour and has been made explicit forming own layers or components in the architecture.
2. Adaptability is realized with feedback loops that can be integrated on different levels into the architecture, for instance, providing global, local, or group feedback.
3. Feedback loops require to monitor and influence the system state, commonly realized in accordance with the MAPE paradigm.

Table 1 Comparison of the properties of reviewed approaches from self-adaptation, self-organization and swarm robotics

Reference [solution name]	Domain	Centralized/ decentralized	Specific focus	Top-down/bottom-up	Approach
Methodologies					
Steghöfer *et al.* (2014) [PosoMAS]	General	Not specified	No	n/a	Software engineering process, simulation
Bonjean *et al.* (2014) [ADELFE]	General	Not specified	No	Bottom-up	Software engineering process, simulation, testing
Morandini *et al.* (2009) [TROPOS4AS]	General	Not specified	No	Mixed	Software engineering process, requirement analysis
Gershenson (2007)	General	Not specified	No	n/a	Software engineering process
Branke *et al.* (2006) [Organic Computing]	General	Hybrid	No	n/a	Learning, feedback-loop architecture, simulation
Edmonds (2005)	General	Not specified	No	n/a	Software engineering process, experimental methods
De Wolf and Holvoet (2007)	General	Not specified	No	n/a	Self-organization algorithm guide
Fernandez-Marquez *et al.* (2012)	General	Not specified	No	n/a	Self-organization algorithm guide, including combinations
Noël and Zambonelli (2015)	General	Not specified	No	n/a	Component decomposition
Middlewares and frameworks					
Hernandez-Sosa *et al.* (2005) [CoolBot]	Robots	Centralized	QoS	n/a	Component-based framework
Cui *et al.* (2014) [ReFrESH]	Embedded systems	Centralized	Fault-detection	n/a	Special reconfiguration layer, built into RTOS
Garlan *et al.* (2004) [Rainbow]	General	Centralized	No	n/a	Separated adaption layer
Sudeikat *et al.* (2009) [SodekoVS]	General	Decentralized	No	n/a	Separated adaption layer, domain specific language
Preisler *et al.* (2013) [Jadex]	General	Decentralized	No	n/a	Separated adaption layer, domain specific language
Graff *et al.* (2014) [jSwarm]	Robot swarms	Centralized	Performance, scheduling	Mixed	Precomputed dependency graph spatial-temporal constraints
Mamei *et al.* (2005) [TOTA]	Sensor networks	Decentralized	Information propagation	Bottom-up	Biological morphogen gradients
Yu and Nagpal (2009)	Modular robots	Centralized	No	Bottom-up	Control laws
Viroli *et al.* (2009) [TuCSoN]	General	Decentralized	No	Bottom-up	Probabilistic and timed co-ordination rules on tuples
Fernandez-Marquez *et al.* (2011) [BIO-CORE]	General	Decentralized	Information propagation	Mixed	Shared data space, pattern services
Pinciroli and Beltrame (2016) [BUZZ]	Robot swarms	Centralized	No	Mixed	Domain specific language, task assignment
Mechanism design and implementation					
Das *et al.* (1995)	Cellular automata	Centralized	Synchronization	Top-down	Genetic algorithms
Matarić (1995)	Robot teams	Decentralized	Motion	Mixed	Set of primitive behaviours, reinforcement function
Klavins (2004) [CCL]	General	Decentralized	No	Bottom-up	Predicate-based functional language
Beal and Bachrach (2006)	Sensor-networks	Decentralized	No	Bottom-up	Functional language, simulation, VM abstraction

Table 1: (Continued)

Reference [solution name]	Domain	Centralized/ decentralized	Specific focus	Top-down/bottom-up	Approach
Bachrach *et al.* (2008) [Protoswarm]	Robot swarms	Decentralized	Motion	Top-down	Functional language, homogenous swarms
Nicola *et al.* (2015) [SCEL]	General	Centralized	General	Top-down	Formal language, tuple space, model checking
Nagpal (2002) [OSL]	Robot self-assembly	Centralized	Assembly	Top-down	Special origami-inspired language, gradient rules
Joshi *et al.* (2014)	Robot self-assembly	Decentralized	Assembly	Top-down	Offline compilation, special application constraints
Yu and Nagpal (2011)	Modular robots	Decentralized	No	Bottom-up	Generalized distributed consensus
Kloetzer and Belta (2006)	Robot swarms	Centralized	Motion	Top-down	Mean and variance of geometric state
Ashley-Rollman *et al.* (2007) [Meld]	Modular robots	Decentralized	No	Top-down	Logic programming language
De Rosa *et al.* (2008) [LDP]	Modular robots	Centralized	No	Top-down	Reactive predicate language
Brambilla *et al.* (2012) [PRISM]	Robot swarms	Centralized	No	Top-down	Property-driven design, formal language
Montagna *et al.* (2013) [LSA]	Pervasive services	Decentralized	No	Mixed	Predicate rules, basic rules to model mechanisms
Francesca *et al.* (2015) [AutoMoDe]	Robot swarms	Centralized	No	Top-down	Evolutionary programming, simulation, optimization

QoS = quality of service; SCEL = Service Component Ensemble Language; OSL = Origami Shape Language; LDP = locally distributed predicates; LSA = live semantic annotations; RTOS = real time operating system; VM = virtual machine.

4. The development of actual co-ordination mechanisms is only supported on an infrastructure level and specialized solutions using evolutionary approaches, logic or predicate programming having limitations in expressiveness and modelling of the dynamic world.
5. Verification and validation is crucial, but existing solutions that either use logic proofs or simulation have limitations in terms of expressiveness or applicability.

   In that sense, we still face the following open challenges:

   - Automatically selecting or determining adaptation or organization mechanisms based on the mission goals;
   - maintaining complex runtime behaviour with regards to given boundaries and application scope;
   - integrating and combining the mechanisms with higher-level decision-making and planning components.

In order to foster the application of adaptive systems, we see numerous opportunities to combine and to extend existing ideas into an applicable framework, leading to new valuable insights.

To make this work, formal methods are beneficial and required in the first stages—this was, for example, demonstrated by Gershenson (2007). Following this spirit, a high-level declarative programming layer can be used for describing the general behaviour of the system and its goals, allowing to deduce certain guarantees, the generation of initial individual behaviour rule sets, and co-ordination patterns. Since such an approach is limited in its expressiveness, it is going to be accompanied with a layer implementing the low-level behaviours that are able to deal with dynamic environment, temporal constraints, and probabilities. A suitable behaviour representation can be realized with a hybrid approach combining behaviour networks with symbolic planning, allowing to decouple goal-directedness and flexible dynamic state transitions, as discussed in Section 7. The configuration of the behaviour sets can be determined through meta heuristic, like evolutionary algorithms, learning, and simulation, like it is applied in evolutionary robotics. This could also be simplified by automatically applying expert knowledge about the application and combination of existing mechanisms as presented in De Wolf and Holvoet (2007) and Fernandez-Marquez *et al.* (2012). An initial setup selected from such an expert knowledge library could avoid unstable initial states. Such an initial setup can be further refined by a learning component, which adjusts the behaviour and mechanisms through reinforcement in order to deal with new situations in accordance with the goal description.

Another additional extension could help to integrate indicators for unintended behaviour describing the boundaries besides a general goal description. This would enable extended control and avoid malicious behaviour.

Driven by our background and our particular interest in multi-robot systems it is necessary to equip the framework with the capability to integrate with existing infrastructures as well as with a convenient abstraction from common sensor and actuator interfaces. For this reason the integration into a widespread framework, like the ROS (Quigley *et al.*, 2009b) appears to be a promising approach.

9 Conclusion

In this work we analyzed and compared existing ideas and approaches in the field of self-adaptation, self-organization, emergent systems, and robotics. The aim of this survey was to find ways, mechanisms, and approaches that simplify the development of systems that comprise large numbers of constituting entities or agents in complex and uncertain environments and to make these systems more robust and to increase their ability to adapt. In this work, we explicitly focused on one particular application domain, namely multi-robot systems. We highlighted unsolved challenges, like an application-independent determination of the self-organization mechanism, validation and control of the system, as well as the integration into the existing decision-making and planning capabilities of intelligent robots. Nevertheless, the presented challenges are also relevant beyond the field of multi-robot systems because, especially creating a link between macro and mirco levels is also important in the multi-agent, self-organizing software and complex systems communities.

In order to accomplish the open challenges, we describe initial draft of a combination of approaches from literature, blended with further extensions and new ideas.

In the future we aim to further detail and expand this draft, for example, by explicitly considering information propagation among the agents. Moreover, we continue the work on a particular framework implementation that directly addresses the future application domain of mobile multi-robots.

Acknowledgement

The authors thank all reviewers and editors for comments that greatly improved the manuscript.

References

Ali, S. M., Zimmer, R. M., Elstob, C. M. & Dubois, D. M. 1998. The question concerning emergence: implications for artificiality. *AIP Conference Proceedings* **437**(1), 138–156.

Allgeuer, P. & Behnke, S. 2013. Hierarchical and state-based architectures for robot behavior planning and control. In *Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots*.

Ashley-Rollman, M., Goldstein, S., Lee, P., Mowry, T. & Pillai, P. 2007. Meld: a declarative approach to programming ensembles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,IROS*, 2794–2800.

Ay, N., Der, R. & Prokopenko, M. 2012. Guided self-organization: perception-action loops of embodied systems. *Theory in Biosciences* **131**(3), 125–127.

Bachrach, J., McLurkin, J. & Grue, A. 2008. Protoswarm: A language for programming multi-robot systems using the amorphous medium abstraction. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, 1175–1178. AAMAS '08. International Foundation for Autonomous Agents and Multiagent Systems.

Balch, T. & Arkin, R. C. 1998. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation* **14**(6), 926–939.

Bashyal, S. & Venayagamoorthy, G. 2008. Human swarm interaction for radiation source search and localization. In *IEEE Swarm Intelligence Symposium, 2008*, 1–8. SIS 2008.

Beal, J. & Bachrach, J. 2006. Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems* **21**(2), 10–19.

Blum, A. L. & Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* **90**(1–2), 281–300.

Bohren, J. & Cousins, S. 2010. The SMACH high-level executive [ros news]. *Robotics Automation Magazine, IEEE* **17**(4), 18–20.

Bonabeau, E., Dorigo, M. & Theraulaz, G. 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press Inc.

Bonet, B. & Geffner, H. 2001. Heuristic search planner 2.0. *AI Magazine* **22**(3), 77.

Bonjean, N., Mefteh, W., Gleizes, M. P., Maurel, C. & Migeon, F. 2014. Adelfe 2.0. In *Handbook on Agent-Oriented Design Processes*, Cossentino, M., Hilaire, V., Molesini, A. & Seidita, V. (eds). Springer, 19–63.

Brambilla, M., Ferrante, E., Birattari, M. & Dorigo, M. 2013. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence* **7**(1), 1–41.

Brambilla, M., Pinciroli, C., Birattari, M. & Dorigo, M. 2012. Property-driven design for swarm robotics. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 139–146.

Branke, J., Mnif, M., Muller-Schloer, C. & Prothmann, H. 2006. Organic computing – addressing complexity by controlled self-organization. In *2th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation ISoLA*, 185–191. IEEE.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. 2004. Tropos: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems* **8**(3), 203–236.

Breuer, T., Giorgana Macedo, G. R., Hartanto, R., Hochgeschwender, N., Holz, D., Hegger, F., Jin, Z., Müller, C., Paulus, J., Reckhaus, M., Álvarez Ruiz, J. A., Plöger, P. G. & Kraetzschmar, G. K. 2012. Johnny: an autonomous service robot for domestic environments. *Journal of Intelligent & Robotic Systems* **66**(1), 245–272.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* **2** (1), 14–23.

Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H, Pezzè, M. & Shaw, M. 2009. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*, 48–70. Springer.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G. & Bonabeau, E. 2003. *Self-Organization in Biological Systems*. Princeton University Press.

CogniTeam Ltd 2015. Cognitao (think as one). `http://www.cogniteam.com/cognitao.html`.

Cui, Y., Voyles, R., Lane, J. & Mahoor, M. 2014. ReFrESH: a self-adaptation framework to support fault tolerance in field mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1576–1582.

Das, R., Crutchfield, J. P., Mitchell, M. & Hanson, James E. 1995. Evolving globally synchronized cellular automata. In *Proceedings of the 6th International Conference on Genetic Algorithms*, 336–343. Morgan Kaufmann Publishers Inc.

De Nicola, R., Ferrari, G., Loreti, M. & Pugliese, R. 2013. A language-based approach to autonomic computing. In *Formal Methods for Components and Objects*, Beckert, B., Damiani, F., de Boer, F. & Bonsangue, M., (eds), Lecture Notes in Computer Science, **7542**. Springer, 25–48.

De Rosa, M., Goldstein, S., Lee, P., Pillai, P. & Campbell, J. 2008. Programming modular robots with locally distributed predicates. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3156–3162.

De Wolf, T. & Holvoet, T. 2005. Emergence versus self-organisation: different concepts but promising when combined. In *Engineering Self-Organising Systems*, Brueckner, S. A., Marzo Serugendo, G., Karageorgos, A. & Nagpal, R. (eds). Springer-Verlag, 1–15.

De Wolf, T. & Holvoet, T. 2007. Design patterns for decentralised coordination in self-organising emergent systems. In *Engineering Self-Organising Systems*, 28–49. Springer.

De Wolf, T., Samaey, G. & Holvoet, T. 2005. Engineering self-organising emergent systems with simulation-based scientific analysis. In *4th International Workshop on Engineering Self-Organising Applications*, 146–160.

Deneubourg, J. L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C. & Chrétien, L. 1990. The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, 356–363. MIT Press.

Desai, J. P., Ostrowski, J. P. & Kumar, V. 2001. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation* **17**(6), 905–908.

Dorigo, M., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Martinez Gonzales, J., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M, O'Grady, R., Pinciroli, C., Pini, G., Retornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stutzle, T., Trianni, V., Tuci, E., Turgut, A. & Vaussard, F. 2013. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics Automation Magazine* **20**(4), 60–71.

Dorigo, M., Trianni, V., Şahin, E., Groβ, R., Labella, T. H, Baldassarre, G., Nolfi, S., Deneubourg, J.-L., Mondada, F., Floreano, D. & Gambardella, L. M. 2004. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots* **17**(2–3), 223–245.

Ducatelle, F., Di Caro, G. A. & Gambardella, L. M. 2010. Cooperative self-organization in a heterogeneous swarm robotic system. In *12th Annual Conference on Genetic and Evolutionary Computation*, GECCO'10, 87–94. ACM.

Edmonds, B. 2005. Using the experimental method to produce reliable self-organised systems. In Brueckner, S. A, Di Marzo Serugendo, G., Karageorgos, A. & Nagpal, R. (eds) *Engineering Self-Organising Systems: Methodologies and Applications*. Springer, 84–99.

Edmonds, B. & Bryson, J. J. 2004. The insufficiency of formal design methods " The Necessity of an Experimental Approach - for the Understanding and Control of Complex MAS". In *3th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, 938–945. IEEE Computer Society.

Fernandez-Marquez, J. L., Serugendo, G. D. M. & Montagna, S. 2011. Bio-core: bio-inspired self-organising mechanisms core. In *International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 59–72. Springer.

Fernandez-Marquez, J. L., Serugendo, G. D. M., Montagna, S., Viroli, M. & Arcos, J. L. 2012. Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing* **12**(1), 43–67.

Fikes, R. E. & Nilsson, N. J. 1972. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2**(3), 189–208.

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Mascia, F., Trianni, V. & Birattari, M. 2015. AutoMoDe-Chocolate: automatic design of control software for robot swarms. *Swarm Intelligence* **9**(2–3), 125–152.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B. & Steenkiste, P. 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer* **37**(10), 46–54.

Gershenson, C. 2007. *Design and control of self-organizing systems*. PhD thesis, Vrije Universiteit Brussel.

Goebel, P. 2013. *ROS By Example*. Lulu.

Goldstein, J. 1999. Emergence as a construct: history and issues. *Emergence* **1**(1), 49–72.

Goodrich, M. A., Pendleton, B., Sujit, P. B. & Pinto, J. 2011. Toward human interaction with bio-inspired robot teams. In *2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2859–2864. IEEE.

Graff, D., Richling, J. & Werner, M. 2013. Programming and managing the swarm - an operating system for an emerging system of mobile devices. In *IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, **0**, 9–16. IEEE Computer Society.

Graff, D., Richling, J. & Werner, M. 2014. jSwarm: distributed coordination in robot swarms. In *Proceedings of the International Workshop on Robotic Sensor Networks (RSN)*.

Groß, R., Bonani, M., Mondada, F. & Dorigo, M. 2006. Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics* **22**(6), 1115–1130.

Haken, H. 1984. *The Science of Structure: Synergetics*. Van Nostrand Reinhold.

Hernandez-Sosa, D., Dominguez-Brito, A. C., Guerra-Artal, C. & Cabrera-Gámez, J. 2005. Runtime self-adaptation in a component-based robotic framework. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2700–2705. IEEE.

Hertzberg, J., Jaeger, H., Zimmer, U. & Morignot, P 1998. A framework for plan execution in behavior-based robots. In *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, 8–13. IEEE.

Hoffmann, J. 2001. FF: the fast-forward planning system. *AI Magazine* **22**(3), 57.

Hrabia, C.-E., Masuch, N. & Albayrak, S. 2015. A metrics framework for quantifying autonomy in complex systems. In Müller, P. J., Ketter, W., Kaminka, G., Wagner, G., and Bulling, N. (eds), *Multiagent System Technologies: 13th German Conference, MATES 2015, Cottbus, Germany, September 28–30, 2015, Revised Selected Papers*. Springer International Publishing, 22–41.

Joshi, R. K., Carbone, P., Wang, F. C., Kravets, V. G., Su, Y., Grigorieva, I. V., Wu, H. A., Geim, A. K. & Nair, R. R. 2014. Precise and ultrafast molecular sieving through graphene oxide membranes. *Science* **343**(6172), 752–754.

Jung, D. 1998. *An architecture for cooperation among autonomous agents*. PhD thesis, University of South Australia.

Kaminka, G. A. 2012. Autonomous agents research in robotics: a report from the trenches. In *AAAI Spring Symposium: Designing Intelligent Robots*.

Kauffman, S. A. 1995. *At Home in the Universe: The Search for Laws of Self-Organization and Complexity*. Oxford University Press.

Kazadi, S. T. 2000. *Swarm engineering*. Phd, California Institute of Technology.

Kephart, J. O. & Chess, D. M. 2003. The vision of autonomic computing. *Computer* **36**(1), 41–50.

Kernbach, S., Meister, E., Schlachter, F., Jebens, K., Szymanski, M., Liedke, J., Laneri, D., Winkler, L., Schmickl, T., Thenius, R., Corradi, P. & Ricotti, L. 2008. Symbiotic Robot Organisms: REPLICATOR and SYMBRION Projects. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, 62–69. ACM.

Klavins, E. 2004. A language for modeling and programming cooperative control systems. In *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04.*, **4**, 3403–3410. IEEE.

Kloetzer, M. & Belta, C. 2006. Hierarchical abstractions for robotic swarms. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation. ICRA 2006*, 952–957.

Krupke, D., Ernestus, M., Hemmer, M. & Fekete, S. 2015. Distributed cohesive control for robot swarms: maintaining good connectivity in the presence of exterior forces. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 413–420.

Lee, Y.-S. & Cho, S.-B. 2014. A hybrid system of hierarchical planning of behaviour selection networks for mobile robot control. *International Journal of Advanced Robotic Systems* **11**(4), 1–13.

de Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K. M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D. B., Sousa, J. P., Tahvildari, L., Wong, K. & Wuttke, J. 2013. Software engineering for self-adaptive systems: a second research roadmap. In *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, de Lemos, R., Giese, H., Müller, H. & Shaw, M. (eds). Springer Verlag, 1–32.

Maes, P. 1989. How to do the right thing. *Connection Science* **1**(3), 291–323.

Mamei, M., Vasirani, M. & Zambonelli, F. 2005. Self-organizing spatial shapes in mobile particles: The TOTA approach. In *Engineering Self-Organising Systems*, 138–153. Springer.

Masar, M. 2013. A biologically inspired swarm robot coordination algorithm for exploration and surveillance. In *IEEE 17th International Conference on Intelligent Engineering Systems (INES)*, 271–275.

Matarić, M. J. 1995. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems* **16**(2–4), 321–331.

Montagna, S., Viroli, M., Fernandez-Marquez, J. L., Di Marzo Serugendo, G. & Zambonelli, F. 2013. Injecting self-organisation into pervasive service ecosystems. *Mobile Networks and Applications* **18**(3), 398–412.

Morandini, M., Migeon, F., Gleizes, M.-P., Maurel, C., Penserini, L. & Perini, A. 2009. A goal-oriented approach for modelling self-organising MAS. In *Engineering Societies in the Agents World X*, Lecture Notes in Computer Science. Springer, 33–48.

Naghsh, A. M., Gancet, J., Tanoto, A. & Roast, C. 2008. Analysis and design of human-robot swarm interaction in firefighting. In *The 17th IEEE International Symposium on Robot and Human Interactive Communication, 2008. RO-MAN 2008*, 255–260. IEEE.

Nagpal, R. 2002. Programmable self-assembly using biologically-inspired multiagent control. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, 418–425. ACM.

Newman, D. V. 1996. Emergence and strange attractors. *Philosophy of Science* **63**(2), 245–261.

Nicola, R. D., Latella, D., Lafuente, A. L., Loreti, M., Margheri, A., Massink, M., Morichetta, A., Pugliese, R., Tiezzi, F. & Vandin, A. 2015. The SCEL language: design, implementation, verification. In *Software Engineering for Collective Autonomic Systems*, Wirsing, M., Holzl, M., Koch, N. & Mayer, P. (eds), Lecture Notes in Computer Science **8998**. Springer International Publishing, 3–71.

Nicolis, G. 1989. Physics of far-from-equilibrium systems and self-organisation. *The New Physics* **11**, 316–347.

Noël, V. & Zambonelli, F. 2015. Methodological guidelines for engineering self-organization and emergence. *Software Engineering for Collective Autonomic Systems*, In Wirsing, M., Huolzl, M., Koch, N. & Mayer, P. (eds), Lecture Notes in Computer Science **8998**. Springer International Publishing, 355–378.

Nolfi, S. & Floreano, D. 2000. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press Cambridge.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S. & Wolf, A. L. 1999. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems* **14**(3), 54–62.

Penders, J., Alboul, L., Witkowski, U., Naghsh, A., Saez-Pons, J., Herbrechtsmeier, S. & El-Habbal, M. 2011. A Robot Swarm Assisting a Human Fire-Fighter. *Advanced Robotics* **25**(1–2), 93–117.

Picard, G. & Gleizes, M.-P. 2004. The ADELFE methodology. In *Methodologies and Software Engineering for Agent Systems*, 157–175. Springer.

Pinciroli, C. & Beltrame, G. 2016. Buzz: An extensible programming language for heterogeneous swarm robotics. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3794–3800. IEEE.

Preisler, T., Vilenica, A. & Renz, W. 2013. Decentralized coordination in self-organizing systems based on peer-to-peer coordination spaces. *Electronic Communications of the EASST* **56**, 1–13.

Prokopenko, M. 2009. Guided self-organization. *Human Frontiers Science Program (HFSP) Journal* **3**(5), 287–289.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. & Ng, A. Y. 2009a. Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software* **3** (3.2): 5.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. & Ng, A. Y. 2009a. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software* **3**, 5.

Reina, A., Dorigo, M. & Trianni, V. 2014. Towards a cognitive design pattern for collective decision-making. In *Swarm Intelligence*, 194–205. Springer International Publishing.

Ren, W. & Beard, A. W. 2004. A decentralized scheme for spacecraft formation flying via the virtual structure approach. *AIAA Journal of Guidance, Control, and Dynamics* **27**, 73–82.

Şahin, E 2005. Swarm robotics: from sources of inspiration to domains of application. In *Swarm Robotics*, Şahin, E. & Spears, W. M. (eds), Lecture Notes in Computer Science **3342**. Springer, 10–20.

Schmeck, H. 2005. Organic computing – a new vision for distributed embedded systems. In *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, 201–203.

Serugendo, G. D. M., Gleizes, M.-P. & Karageorgos, A. 2005. Self-organization in multi-agent systems. *The Knowledge Engineering Review* **20**(2), 165–189.

Serugendo, G. D. M., Gleizes, M. P. & Karageorgos, A. 2006. Self-organisation and emergence in MAS: an overview. *Informatica (Slovenia)* **30**(1), 45–54.

Steghöfer, J.-P., Seebach, H., Eberhardinger, B. & Reif, W. 2014. PosoMAS: an extensible, modular SE process for open self-organising systems. In *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, Dam, H. K., Pitt, J., Xu, Y., Governatori, G. & Ito, T. (eds), Lecture Notes in Computer Science **8861**. Springer International Publishing, 1–17.

Sudeikat, J., Braubach, L., Pokahr, A., Renz, W. & Lamersdorf, W. 2009. Systematically engineering self-organizing systems: the SodekoVS approach. *Electronic Communications of the EASST* **17**, 1–12.

Sudeikat, J. & Renz, W. 2009. MASDynamics: toward systemic modeling of decentralized agent coordination. In *Kommunikation in Verteilten Systemen (KiVS), Informatik aktuell*, David, K. & Geihs, K. (eds). Springer, 79–90.

Viroli, M., Casadei, M. & Omicini, A. 2009. A framework for modelling and implementing self-organising coordination. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, 1353–1360. ACM.

Viscido, S. V., Parrish, J. K. & Grünbaum, D. 2004. Individual behavior and emergent properties of fish schools: a comparison of observation and theory. *Marine Ecology Progress Series* **273**, 239–250.

Wegner, P. 1997. Why interaction is more powerful than algorithms. *Communications of the ACM* **40**(5), 80–91.

Winfield, A. F. T., Harper, C. J. & Nembrini, J. 2004. Towards dependable swarms and a new discipline of swarm engineering. In *Swarm Robotics*, 126–142. Springer.

Ye, D., Zhang, M. & Vasilakos, A. V. 2017. A survey of self-organization mechanisms in multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **47**(3), 441–461.

Yu, C.-H. & Nagpal, R. 2009. Self-adapting modular robotics: A generalized distributed consensus framework. In *IEEE International Conference on Robotics and Automation(ICRA)*, 1881–1888. IEEE.

Yu, C.-H. & Nagpal, R. 2011. A self-adaptive framework for modular robots in a dynamic environment: theory and applications. *The International Journal of Robotics Research* **30**(8), 1015–1036.