

## Towards Self-organizing Swarms of Reconfigurable Self-aware Robots

Oliver Kosak, Constantin Wanninger, Andreas Angerer, Alwin Hoffmann,  
Alexander Schiendorfer, and Hella Seebach  
*Institute for Software & Systems Engineering, University of Augsburg, Germany*  
*E-Mail: {kosak, wanninger, angerer, hoffmann, schiendorfer, seebach}@isse.de*

**Abstract**—Designing complex adaptive systems for real world applications is a delicate challenge, especially when support for humans in crucial situations should be achieved. In this position paper, we propose a multi-agent based approach for physically reconfigurable, heterogeneous robot swarms. These can be deployed when there is a need to search, continuously observe and react, e.g. in disaster scenarios. We show first results that validate the feasibility of our approach.

**Index Terms**—Multi-Agent Systems; Sensor Systems; Unmanned Aerial Vehicles; Self Awareness; Robot Swarms

### 1. Introduction

Heterogeneous collective systems are becoming increasingly popular (cf. [1], [2]), as it is appealing to combine the strengths of different devices – especially for *Search and Rescue* missions (cf. [3], [4], [5]). This problem class uses collective systems of multiple, mostly flying robots to simultaneously search in different areas to reduce the time until, e.g., missing persons are found. Another use of multiple flying robots is the *Observation of Critical Infrastructure* such as pipelines, railways, or wind turbines (cf. [6]). Here, the area to cover is predefined and the collective system has to simultaneously observe this known area. In each of the abovementioned approaches, the system has to achieve a small set of predefined tasks – in some cases, it has even been designed exclusively for a certain task.

From our point of view, there is great potential for collective multi-robot systems far beyond these application areas. We have identified the application class of *Search, continuously Observe and React (ScORe)* missions which we consider a generalization of search and rescue missions [3]. ScORe missions are characterized by the following properties : (i) search multiple a priori unknown parameters; (ii) continuously observe and track the relevant parameters; (iii) both in a wide area; (iv) perform collective on-line evaluation of gathered data; and (v) trigger reactions by user input or due to abnormalities or patterns in evaluated data.

ScORe missions are necessary e.g. in chemical and nuclear accidents, which is sketched in detail in Sect. 2. Another scenario where ScORe missions come into play is the in-situ measurement of interacting climate parameters (such as temperature, humidity and green house gas distribution)

where it is important to continuously gather consistent data at different hot spots over a long period.

A multi-robot swarm that should accomplish a ScORe mission faces several challenges that are to a large extent antagonistic. The *operation time* of the swarm as a whole should be maximized (cf. ScORe properties (ii) and (iii)). Two ways to achieve this are reducing the power consumption of the individual robots and adding more robots to introduce redundancy. On the other hand, the *swarm size* in terms of individual robots should be kept low for economical and usability reasons: more robots are of course more expensive and also more complex to transport and set up. Thus, a way to increase the *versatility* of the swarm (cf. ScORe properties (i) and (v)) is to equip individual robots with more sensing and actuating capabilities, which has negative effects on power consumption and thus operation time in turn. This position paper proposes to introduce physical reconfigurability on hardware level and integrate this new ability in the design of a collective adaptive system. In particular, we propose an architecture that provides the following contributions: (a) enable physical reconfigurability using knowledge-based self-awareness of hardware modules; (b) allow for interwoven optimization of task execution and adaptation of the system’s hardware configuration to the current task(s); (c) provide users with a high-level, domain-specific and extensible task abstraction for controlling the system; and (d) support simultaneous execution of multiple tasks issued at runtime of the system.

In the next section, we give a ScORe mission example that is used throughout the paper. Sect. 3 introduces the main idea of our approach. In Sect. 4, we propose a generic software architecture for robot swarms as sketched above. Sect. 5 presents results of a first evaluation. Finally, a conclusion and an outlook is given in Sect. 6.

### 2. Case Study

As illustrative example, consider a chemical accident where fire fighters face the threat of toxic gases. The challenges in such a case are manifold. First of all, fire fighters need to identify one or more hazardous gases that constitute the gas cloud and measure their concentrations. This can be feasible in road or train accidents, but very difficult, e.g., in synthetic material plants. Subsequently, they must determine the current dimensions of the gas cloud, and more important

predict its expansion and relevant concentrations. This is necessary to identify endangered residential areas and to evacuate the population.

Currently, fire fighters must make these observations manually and take actions with limited, incomplete or even wrong information [7]. For example, measurements of gas concentrations are mainly done at the location where the accident happened and a few other points close to the ground. Based on these measurements, gas types and concentrations are identified and the dimensions of the gas cloud are roughly estimated. Moreover, the expansion of the gas cloud is predicted based on available data about local weather conditions. However, this is difficult or imprecise, e.g., due to few ground measurements, different expansion patterns of gases, outdated weather data or not correctly identified hazardous gases. Based on those rough and deficient estimations, extensive decisions have to be made whether to evacuate areas of residential houses, retirement homes or even hospitals. To evacuate a certain area, fire fighters drive around making loudspeaker announcements to the population. As far as staff is available, they also ring doorbells – particularly in large buildings. Still, it is not easy to reach everyone in rural areas or urban recreation areas where, e.g., people are out for walks. In such a case, fire fighters often need to go into endangered areas to help and rescue people exposing themselves to hazardous gases.

A swarm of (flying) robots could substantially improve the way such accidents can be handled as shown by previous work (cf. [7], [8]). There, e.g., a swarm is responsible for *tracking the dimension of the gas cloud and the concentrations of gases*. At the same time, it improves predictions of the gas cloud expansion by *in-situ measurements of weather conditions like temperature, wind direction and wind strength*. Furthermore, flying robots *serve as a communication relay* to establish a robust wireless radio network for fire fighters even under harsh conditions without hindering their other activities (e.g., collision avoidance [9]). We put our focus on physical reconfigurability to handle ScORe missions like these, which has not been considered by previous work.

### 3. Approach

We consider robots that consist of certain *basic components*, e.g., a quadcopter frame including motors, propellers and a flight controller or a mobile robot including motors, wheels and a navigation controller. In order to be functional, those basic components must be equipped with a *power source*, usually a battery. We assume that, given a power source, the basic components exhibit certain basic functionality, i.e., flying or driving to a given point in space.

The robots may be extended with so-called *cube modules* containing a microprocessor and additional sensors or actuators. Each of these modules contains self descriptive information pertaining to the individual capabilities of the module, which allows access to e.g. measurements without prior knowledge of the sensor in question. Retaining the correlation between sensor data and its meaning, combined

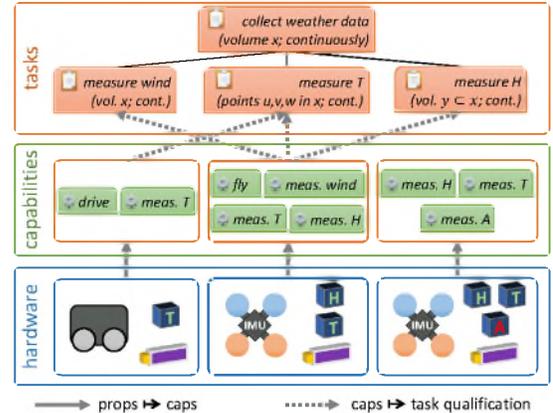


Figure 1. Three different device configurations, respective capabilities and the resulting task qualification. Each quadcopter has built-in sensors (i.e., the inertial measurement unit) that are required for flying, but may also be employed to estimate quantities like wind speed and direction during flight.

with the additional computational power in each module, offer great improvement over previous approaches, as e.g. the Cubelets project [10]. Physical reconfigurability can be performed by interchanging standardized cube modules.

In our case study, e.g., the swarm can start with a broad variety of gas sensors. After identifying the type of the toxic gas, it exchanges its sensors to only track the relevant gas. Hence, a quadrotor can carry additional batteries, sensors to record weather data (e.g., for temperature, or humidity), or cameras for *finding missing persons*. In the sudden case of an evacuation, robots could be *equipped with loudspeakers* as actuators, increasing flexibility and coverage of announcements to the population. Finally, if necessary, swarm members with grippers can *transport light equipment such as gas masks or communication devices to endangered persons*.

We assume all components to be self-aware to allow the system to reason about its current physical configuration. We use the term *properties* to denote all known physical features of a component (e.g., its weight) as well as specific features (e.g., battery capacity or sensor resolution). Fig. 1 shows three differently configured robots. The left robot is a mobile ground robot equipped with power source and one cube module with an air temperature sensor (T). Important properties include the *maximum payload* of this mobile robot, the *weight* of each component, the battery *capacity* and the *sensor type* contained in the cube module. The quadcopter in the bottom center of the figure is equipped with two cube modules T and H, where H contains an air humidity sensor. The built-in IMU is a further sensor of the quadcopter. Finally, the second quadcopter (bottom right) is equipped with an additional cube module A that contains a gas sensor. The combination of basic components and further cube modules that add functionality to a robot distinguishes our approach from existing work regarding self-assembling and self-reconfiguring robotic systems (cf. [11], [12]). We believe that our more coarse-grained approach allows for easier reasoning about the abilities of a robot depending on

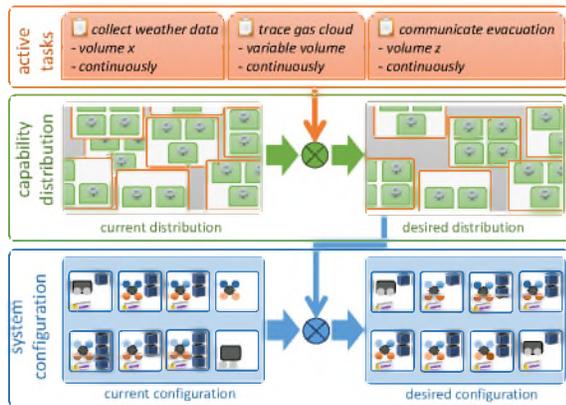


Figure 2. Self-organized system reconfiguration to optimize multi-task fulfillment. After identifying the desired capability distribution for one part of the swarm, this part reconfigures its hardware components accordingly.

its current configuration, as outlined below.

According to its physical configuration, a robot can have different *capabilities*. A capability models the robot’s ability to perform certain operations, e.g., driving, flying or measuring a certain quantity. Fig. 1 also depicts the capabilities of the three robots explained above. The deduction of those capabilities is denoted by the mapping  $props \mapsto caps$ . This mapping is non-trivial and depends on the properties of all modules involved. For example, the left robot has the capabilities *drive* and *measure T*, whereas the right robot neither has the capability to fly nor to measure wind during flight due to being overloaded.

The behavior of the collective system is defined by the *tasks* it should achieve. Such tasks can be given at runtime by users or emerge from inside the system itself. They may be domain-specific and, moreover, may be composed of sub-tasks. Some (sub-)tasks can be accomplished by single individuals configured in certain ways, while other tasks require cooperation of multiple robots. Fig. 1 shows an example task (i.e., *collect weather data*) that contains multiple sub-tasks (e.g., *measure wind*). Tasks contain a specification that states constraints for task fulfillment, like continuous operation for a certain time in a certain area or at predefined points of interest. Deciding which robot can contribute to which task requires another mapping  $caps \mapsto task\ qualification$  that depends on the capabilities of the robot. Consider the left and middle robot in Fig. 1. Both can contribute to fulfilling the sub-task *measure T* for points u,v,w near the ground, whereas only the middle robot can contribute to *measure wind* and *measure H* as well. The right robot will not be able to contribute to any task, as it is not able to take off and change its position accordingly.

For the intended class of applications (i.e., ScORe missions), the system in most cases should fulfill multiple tasks at the same time as depicted in Fig. 2 with three currently available tasks. As discussed before, many robots will only be able to contribute to some of the available tasks, and there might even be tasks for which no robot is configured such that it can contribute to those tasks. Thus, the system should

combine self-organization principles and physical reconfigurability to optimize itself in order to fulfill the current tasks. To achieve this, we propose a two-stage approach: First, the distribution of currently available capabilities should be analyzed and optimized depending on the current tasks on hand (cf. Fig. 2). To determine the current capability distribution throughout the swarm, the system has to employ mechanisms to cope with limited connectivity among its individuals. Based on the desired capability distribution and the current physical configuration, a new desired physical configuration is calculated. This calculation should also consider the effort involved for applying the reconfiguration. For a quadcopter, e.g., this might involve landing at a certain place and having its cube modules replaced by the user or an automated system. The same mechanism for optimizing the system configuration may also be employed to integrate new robots into the system (cf. the *empty* quadcopter and mobile robot in the bottom left part of Fig. 2) and decide about their *best* physical configuration. In comparison to the taxonomy of multi-robot task allocation given in [13], the possibility for physical reconfiguration (e.g., dynamically changing robot capabilities) further increases the complexity of relations between robots and tasks. In general, well suited planning mechanisms have to be found, that enable the highly dynamic and distributed system to achieve the user-defined tasks appropriately. As numerous literature on these topics already exists (cf., [13], [14]), we are certain to overcome this challenge.

## 4. Architecture

To empower a system with self-organizing properties mentioned above, we propose a Multi-Agent system (MAS) based architecture as presented in Fig. 3. We distinguish between *User Devices (UDs)* and *Self-organizing Devices (SoDs)*. The latter, as a whole, represent the autonomous part of the system. Each SoD is responsible for controlling the robot it is detached to as well as for interacting with other devices in the system. As functionality is encapsulated in software agents, each SoD is required to run multiple agents at once, i.e., to provide a MAS platform on its own (we use the Jadex Active Components Framework [15]). The system as a whole, therefore, can be described as a System-of-Systems [16]. As great advantage over other known approaches (cf., [3], [5]) our architecture intends to be constructed for allowing to switch between completely different tasks without having to modify each SoD’s MAS. Modifications only have to be made to the knowledge base of available capabilities as well as to the task definition, that defines capabilities needed for its solving.

**User Agent.** The UD offers an interface for humans to interact with SoDs. This functionality is encapsulated in a *User Agent* running on a MAS platform compatible with those on the SoDs. By complying with the MAS-specific communication protocol, the User Agent is able to transfer user-defined tasks to SoDs as well as to observe the execution and status of already submitted tasks. A task’s status can be *unsolved*, *requested*, *active*, or *solved*. Additionally, how

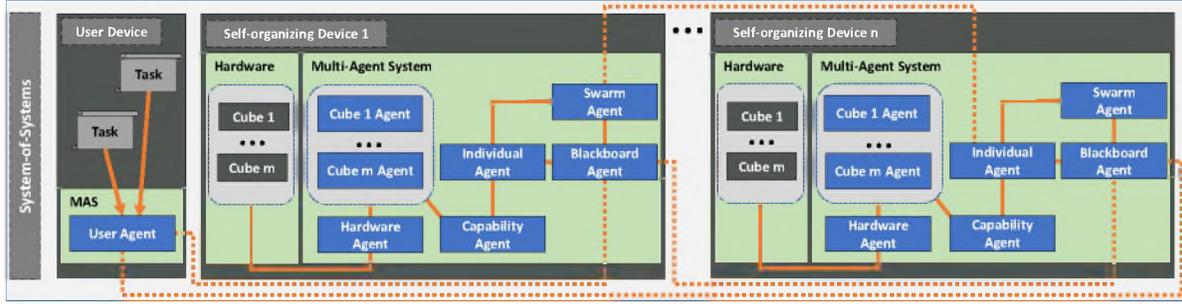


Figure 3. System-of-Systems architecture: Each of the  $n$  *Self-organizing Devices* (SoD) provides the platform for an equal set of agents that form a *Multi-Agent System* running on the device. In addition to this, a *User Device* represents the instructional part, offering an interface to a human operator for introducing new tasks. Solid lines indicate intra-device communication, dotted lines inter-device (wireless) communication for a scenario where the *Swarm Coordination Agent* on SoD<sub>1</sub> coordinates  $n$  *Individual Coordination Agents* on its local device as well as on remote devices. Each currently attached hardware component is abstracted by the concept of a *Cube* that is represented by a software agent (*Cube Agent*) in the corresponding SoD’s MAS.

a task can be solved is defined by a dependency graph of needed capabilities. New user-defined tasks are classified as *unsolved* and transmitted from the User Agent to reachable *Blackboard Agents*, located on every SoD. User-defined tasks include possibilities for task decomposition. These initial execution plans can be calculated by the User Agent by using methods like those described in [13].

**Blackboard Agent.** Hence, Blackboard Agents have the responsibility to distribute information about new tasks and their status on an *inter-device level* by communicating with other reachable Blackboard Agents. This is necessary to reduce the risk of losing information about tasks – albeit there are no guarantees especially in dynamic wireless communication networks [17] – as information loss could reduce the system performance significantly (e.g., tasks are re-executed although they were already solved, or tasks are not executed at all). Therefore, Blackboard Agents gossip relevant information as soon as their information status has changed. Besides, the User Agent is included in this information distribution process to be able to inform the user about task execution and status as mentioned above. Furthermore, a Blackboard Agent communicates on an *intra-device level* with an *Individual Agent* and a *Swarm Agent* located on the same SoD. While the former classifies its SoD’s current qualification for each task, the latter tries to find a cooperation structure for solving a task if necessary.

**Capability Agent.** To determine the current qualification, the Individual Agent relies on the information originating from the SoD’s *Capability Agent* (cf. Fig. 3). To retrieve the current capabilities, this agent interacts with *Cube Agents* that are currently running on the SoD. By being an abstraction from low-level hardware protocols and implementation, they represent the properties of sensors and actuators (i.e., the *Cubes*) as software agents and offer an interface for capability execution. As capabilities may not only be directly associated to properties provided by hardware<sup>1</sup>, the Capability Agent has access to a knowledge

1. The capability *flying*, e.g., may be lost as another cube with high weight is connected, or the capability *measure current height* becomes available as cubes providing temperature and pressure measurements are connected.

base for deducing which capabilities can be made available.

**Cube and Hardware Agents.** To ensure that each connected Cube is represented through an agent, the *Hardware Agent* dynamically starts and stops Cube Agents as the configuration of its SoD changes. In addition, the Hardware Agent is also responsible for coordinating all hardware-specific operations including the access on communication buses or the addressing of underlying hardware.

**Individual Agent.** For every task, the Individual Agent compares the Capability Agent’s information to the task’s capability dependency graph. To decide the SoD’s qualification for a task, we propose to use constraint solving techniques. More specifically, the agent minimizes the number of required Cube changes, such that the device is suited for the task afterwards. If no change is needed, the current configuration is already an optimal solution; otherwise a suggestion involving a minimal number of reconfigurations is returned. In fact, this formulation can be encoded as a soft constraint problem [18]. Thus, we have a decision variable  $s_i$  for every slot  $i$  that maps to an available Cube or  $\perp$  if it is empty. For each slot  $i$ , we have a soft constraint ( $s_i^{\text{old}} = s_i$ ) that is violated if we change the Cube attachment. Our objective is to minimize the number of violated soft constraints (i.e., a Max-CSP). Hard constraints regulate that the planned configuration offers all capabilities required for a task. We plan to specify prioritized preferences regarding reconfiguration decisions (e.g., which component exchange is less likely to cause problems) with more precision using *constraint relationships* [19]. For every task solvable by reconfiguration, the Individual Agent creates a *Reconfiguration Request Task* and uses the Blackboard Agent’s information distribution mechanism. Reconfiguration Request Tasks include information about tasks that become solvable through Cube exchanges on an SoD which in turn can be used by Swarm Agents to suggest broader reconfiguration, e.g., for a set of SoDs.

After classifying all tasks, the Individual Agent tries to assign solvable tasks to its SoD. To avoid multiple task assignments, the Blackboard Agent is used again. As every Individual Agent informs its local Blackboard Agent before actually assigning a task, conflicts can easily be handled:

When receiving a task with status *requested*, Blackboard Agents distribute this information in the known manner. In case a further Individual Agent, located on another SoD, also requested the assignment of this task, this can be resolved by any established leader election heuristic for distributed systems.

When an Individual Agent finally has assigned a task, it is also responsible for its correct execution. Therefore, according to the task’s dependency graph, the Individual Agent requests capability execution from its local Capability Agent which further triggers appropriate Cube Agents. Furthermore, the Individual Agent is able to handle coordination requirements that are defined in the tasks description – e.g., sending synchronization messages to a *Swarm Agent* after executing a capability.

**Swarm Agent.** All tasks classified as not solvable by the SoD alone, as well as known Reconfiguration Request Tasks, are passed to the local Swarm Agent (cf. Fig. 3), which tries to solve them. It is capable of transforming the original task into a set of sub-tasks (cf. Sect. 3) containing additional information, that each of them has to be solved in cooperation with the Swarm Agent. These sub-tasks are transmitted to the Blackboard Agent, that distributes them. Individual Agents, capable of executing one of these sub-tasks, handle them the usual way first (i.e., determine the SoD’s task qualification) and subsequently inform the Swarm Agent. When at least one Individual Agent is available for every sub-task, the Swarm Agent activates the task and informs a suitable team of Individual Agents that they now have to perform the sub-tasks cooperatively. The execution of sub-tasks is coordinated by the Swarm Agent, that is able to exchange information with its cooperation partners – which is necessary to meet the tasks Capability dependency graph. With the Swarm Agent taking the leader role, the original task finally can be solved appropriately. Having solved the task, (for continuous tasks: after the task has been deactivated), the Swarm Agent dissolves the team structure and informs the Blackboard Agent.

In addition to normal tasks, a Swarm Agent is able to handle Reconfiguration Request Tasks that describe possible local Cube reconfigurations or such originating from other SoDs. Therefore, it tries to find a new over-all ensemble configuration, enabling the system to solve as many currently unsolvable tasks as possible (cf. Fig. 2). This creates a new *Reconfiguration Recommendation Task* that is distributed via the Blackboard Agent. As Swarm Agents on different SoDs create Reconfiguration Recommendation Tasks, which can be based on different knowledge each as explained above, the best of these solutions can be selected. The actual reconfiguration can be performed when appropriate (e.g., if no task status changed over a defined time interval). Reconfiguration Recommendation Tasks, like every other task, rely on SoDs that are able to solve them. This can be an SoD having the capability to reconfigure other SoDs (e.g., a robot manipulator cube).

With the use of software agents, the proposed collective system is able to autonomously solve user-defined tasks that in general are solvable with the existing set of available

Cubes and SoDs. Hence, there is a solution if the collective system is suitable to solve the task. However, one or more reconfigurations may be necessary.

## 5. Evaluation

To substantiate the presented ideas with feasibility results, we performed a proof-of-concept evaluation on a single board computer (*SBC*) that is capable of running a SoD and can be mounted on robots like small UAVs, as first experiments have shown. In particular, we consider its ability to solve constraint optimization problems as part of the internal reasoning. This evaluation addresses the question whether it is feasible to use a state-of-the-art constraint solver on an SBC for a practical problem. Though this question addresses a relatively narrow part of our complete approach, we consider it nevertheless an important prerequisite for its feasibility.

We choose a well-defined optimization problem, i.e., a single-agent single-task allocation problem where  $n$  tasks have to be assigned to  $n$  agents with given non-negative costs for every agent/task pair. The objective is to minimize the worst costs any agent encounters<sup>2</sup>. Formally:

$$\begin{aligned} & \underset{t_1, \dots, t_n}{\text{minimize}} && \max_{i \in \{1, \dots, n\}} C[i, t_i] && (1) \\ & \text{subject to} && i \neq j \rightarrow t_i \neq t_j \\ & && t_i \in \{1, \dots, n\} \end{aligned}$$

where  $t_i$  maps to the task agent  $i$  performs and  $C[i, j]$  is the cost matrix. The problem’s only parameters are  $n$  and  $C$  which makes it attractive to generate random instances. Specifically, for a problem consisting of  $n$  agents and tasks, we generate random 3D coordinates in a rectangular cuboid with a width and length of 1000m, and a height of 200m to represent a volume typical for a ScORE mission. We implemented both a greedy algorithm that iteratively selects the shortest agent/task pair as well as a MiniZinc [20] model that is optimized using Gecode.<sup>3</sup> The solving behavior on identical problems on a PC and the SBC with a 60s timeout was explored. The problem size  $n$  ranged from 10 to 40, with 15 problems generated for each value of  $n$ . Quantities of interest are the time to find an optimal solution, the ratio of problems optimally solved, and the achieved objectives.

Figure 4 visualizes that the performance gap is rather low. For problems involving 10 or 15 agents, the average runtime ranged between 0.5s and 1s on the SBC, whereas it ranged between 0.03s and 0.15s on the PC. For larger problems, the average difference amounts to few seconds until both platforms converge to the timeout. However, the solving behavior is very similar with the SBC failing to

2. The problem models the assignment of agents to positions in a formation, minimizing the farthest distance any agent has to move. That way, we find allocations that minimize the total duration to establish the formation. Compared to a greedy algorithm, the constraint solver saved a significant amount of flight time with comparatively few invest of calculation time.

3. Source code at <https://git.io/vKeJx> and <http://www.gecode.org/>. Evaluation devices were a PC (4 x 3.2 GHz, 16 GB RAM, Ubuntu 15.10) and an SBC, i.e. an Odroid XU4 (8 x ARM A15/7, 2 GB RAM, Ubuntu 14.04).

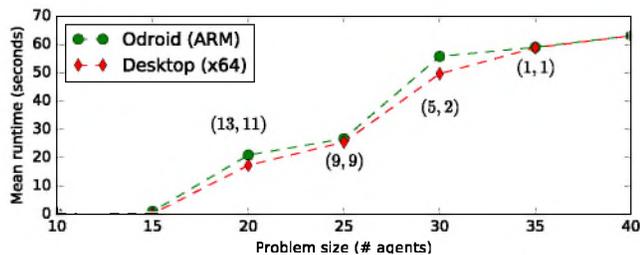


Figure 4. Runtime gap between an x64 PC and an Odroid. Standard deviations are omitted for clarity. Values  $(x, y)$  indicate the number of problems (out of 15) the PC ( $x$ ) and Odroid ( $y$ ) could prove optimal (close). For  $n \in \{10, 15\}$ , all problems were closed, for  $n = 40$ , none was.

prove optimality in only five fewer cases compared to the PC. The overhead of the objective values of the best found solution on an SBC (after 60s) compared to the PC was 2%, averaged over all instances. Further experiments suggest that, beyond  $n = 40$ , it becomes hard to prove optimality in 60s, even for the PC. We can conclude that there is only a small performance gap towards an SBC which seems to have little trouble proving optimality for problems involving up to 20 agents. First experiments also confirm that the selected SBC is not only capable of running a CSOP solver but also to control real robots (UAV and ground robots) with our MAS-based architecture (we use Jadex Active Components [15]) in parallel.

## 6. Conclusion

In this paper, we have introduced the broad application class of ScORe missions which benefits from a collective adaptive system of reconfigurable self-aware robots. In particular, reconfigurability and self-organization can be employed to (1) fulfill different tasks at the same time such as tracking a gas cloud, gathering weather data, or informing people about evacuation, (2) adapt the capabilities of its individuals to optimally suit the current tasks, e.g., optimize the number and distribution of particular sensors, and (3) establish robust continuous operation despite failures and limited battery power. The proposed architecture can achieve these goals and, thus, can be employed to any ScORe mission. Moreover, an evaluation has shown that it is feasible to use a constraint solver for optimization on mobile robots despite very limited resources. Finally, we have performed first experiments controlling real robots with a MAS that have been very promising. Our next steps include improvements concerning generic and extensible task definition. The goal is to enable users of our swarm to control it merely on task level. We will also improve mechanisms for task decomposition and assignment to further increase the autonomy of our system. Further, consensus mechanisms have to be integrated that hinder the system from cyclically reconfiguring its overall structure as we consider reconfiguration to be executed in a decentralized manner.

## References

- [1] A. Prorok, M. A. Hsieh, and V. Kumar, "Fast redistribution of a swarm of heterogeneous robots," in *Proc. 9th EAI Intl. Conf. on Bio-inspired Information & Comm. Technologies (BICT)*, 2015.
- [2] E. F. Flushing, L. M. Gambardella, and G. A. D. Caro, "A mathematical programming approach to collaborative missions with heterogeneous teams," in *Proc. 2014 IEEE/RSJ Intl. Conf. on Intel. Robots & Systems (IROS)*. IEEE, 2014.
- [3] R. R. Murphy *et al.*, "Search and rescue robotics," in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2008, ch. 50.
- [4] J. Scherer *et al.*, "An autonomous multi-uav system for search and rescue," in *Proc. 1st Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet '15)*. ACM, 2015.
- [5] M. Dorigo *et al.*, "Swarmoid: A novel concept for the study of heterogeneous robotic swarms," *IEEE RAM*, vol. 20, no. 4, pp. 60–71, 2013.
- [6] SNCf Réseau. Drones serving the needs of industry.
- [7] K. Daniel, B. Dusza, A. Lewandowski, and C. Wietfeld, "Airshield: A system-of-systems muav remote sensing architecture for disaster response," in *Proc. 3rd Annual IEEE Systems Conf. (SysCon)*, 2009.
- [8] N. Lewyckyj, J. Biesemans, and J. Everaerts, "OSIRIS: A european project using a high altitude platform for forest fire monitoring," in *Safety and Security Engineering II*, ser. WIT Transactions on The Built Environment. WIT Press, 2007, vol. 94, pp. 205–213.
- [9] N. M. Alexandrov and T. A. Ozoroski, "Design for survivability: An approach to assured autonomy," *AAA Aviation*, 2016.
- [10] M. D. Gross and C. Veitch, "Beyond top down: Designing with cubelets," *Tecnologias, Sociedade e Conhecimento*, vol. 1, no. 1, pp. 150–164, 2013.
- [11] M. Yim *et al.*, "Modular self-reconfigurable robot systems," *IEEE RAM*, vol. 14, no. 1, pp. 43–52, 2007.
- [12] H. Li, T. Wang, H. Wei, and C. Meng, "Response strategy to environmental cues for modular robots with self-assembly from swarm to articulated robots," *J. Intell. & Robotic Systems*, vol. 81, no. 3, pp. 359–376, 2015.
- [13] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.
- [14] F. Amigoni, N. Gatti, C. Pinciroli, and M. Roveri, "What planner for ambient intelligence applications?" *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 35, no. 1, pp. 7–21, Jan 2005.
- [15] A. Pokahr, L. Braubach, and K. Jander, "The Jadex project: Programming model," in *Multiagent Systems and Applications*, ser. Intelligent Systems Reference Library, M. Ganzha and C. L. Jain, Eds. Springer, 2013, vol. 45, pp. 21–53.
- [16] S. Mittal *et al.*, "Modeling and simulation for systems of systems engineering," in *Systems of Systems Engineering*, M. Jamshidi, Ed. Wiley, 2008.
- [17] K. P. Birman, "Network perspective," in *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*. Springer, 2012, pp. 101–143.
- [18] P. Meseguer, F. Rossi, and T. Schiex, "Soft Constraints," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, ch. 9.
- [19] A. Schiendorfer *et al.*, "Constraint Relationships for Soft Constraints," in *Proc. 33rd SGAI Int. Conf. Innov. Techniques & Applic. of Artificial Intelligence (AI'13)*. Springer, 2013.
- [20] N. Nethercote *et al.*, "Minizinc: Towards a standard cp modelling language," in *Principles and Practice of Constraint Programming—CP 2007*. Springer, 2007, pp. 529–543.