Swarm and Collective Capabilities for Multipotent Robot Ensembles

Oliver Kosak(✉), Felix Bohn, Lennart Eing, Dennis Rall, Constantin Wanninger, Alwin Hoffmann, and Wolfgang Reif

Institute for Software and Systems Engineering at the University of Augsburg, Universitätsstraße 2, 86159 Augsburg, Germany
kosak@isse.de

Abstract. Swarm behavior can be very beneficial for real-world robot applications. While analyzing the current state of research, we identified that many studied swarm algorithms foremost aim at modifying the movement vector of the executing robot. In this paper, we demonstrate how we encapsulate this behavior in a general pattern that robots can execute with adjusted parameters for realizing different beneficial swarm algorithms. We integrate the pattern as a virtual swarm capability in our reference architecture for multipotent, reconfigurable multi-robot ensembles and demonstrate its application in proof of concepts. We further illustrate how we can lift the concept of virtual capabilities to also integrate other known approaches for collective system programming as virtual collective capabilities. As an example, we do so by integrating the execution platform for the Protelis aggregate programming language.

Keywords: Swarm behavior · Multi-agent systems · Robot swarms · Multipotent systems · Collective adaptive systems · Ensembles

1 Motivation

The use of ensembles or swarms of autonomous robots, especially unmanned aerial vehicles (UAV), is very beneficial in many situations in our daily life. This statement is validated by the multitude of different applications for ensembles that emerged during the past decade making use of the benefits collective behavior can deliver, e.g., with emergent effects achieved by swarm behavior. Unfortunately, the current trend is that every single new application also requires a new software approach for its realization [3,8]. While these specialized approaches show beneficial results for their dedicated applications, e.g., using collective swarm behavior for searching [27], or distributed surveillance [15,16] among many others, users can find it hard to adapt them and profit from previous developments in (even only slightly) different use cases.

To come by this issue, we propose to make use of a *common pattern* instead that can express the collective swarm behavior of a certain class in general. Developers of multi-robot systems can implement such pattern once at design time

Partially funded by DFG (German Research Foundation), grant number 402956354.

and parametrize it differently at run-time to achieve specific emergent effects. We identified such a common pattern researchers frequently use for implementing *movement-vector based swarm behavior* of different types in swarm robotic systems. While producing a different emergent effect each, we can see that swarm algorithms like the particle swarm optimization algorithm [27], the commonly known flocking behavior originally analyzed in [21], shaping and formation algorithms [22], and distribution algorithms [15,16] make use of the same set of local actions: measuring one or multiple specific parameters, communicating with neighbors in the swarm, and modifying the movement vector of the robot.

For this paper, we implement such a common pattern in our reference architecture for multipotent multi-robot ensembles [9,14]. Therefore, we introduce the concept of *configurable, virtual, collective capabilities* that encapsulate complex behavior of individual robots by composing other capabilities, i.e., services a robot already provides, and produce collective behavior when executed cooperatively in an ensemble. For example, to realize flocking behavior following [21], each individual robot requires to execute certain capabilities in an appropriate combination, perform position and velocity measurements, needs to exchange resulting values with swarm members and adapt its movement vector accordingly, which then results in the collective emergent effect of the individuals forming a flock as an ensemble. By executing such a virtual capability collectively in a multi-robot system, we can realize swarm behavior and achieve useful emergent effects. We further validate the concept of virtual collective capabilities by demonstrating how other approaches for programming collective behavior can be integrated into our multipotent systems reference architecture by the example of Protelis [19] as a further example of a virtual collective capability. The contributions of this paper thus are: *1)* The identification and demonstration of a common pattern for realizing swarm behavior for collective adaptive systems, *2)* the extension of our current reference architecture for multipotent systems with the concept of virtual capabilities, *3)* the integration and evaluation of virtual capabilities realizing collective behavior for multipotent systems with our common swarm pattern and the external approach Protelis [19].

The remainder of the paper is structured as follows. In Sect. 2 we illustrate our objectives and highlight the challenges we need to tackle and then propose our solution in Sect. 3. In Sect. 4 we demonstrate the functionality of our approach for our case study in a simulation environment and deliver proof of concepts supported by expressive video materials. In Sect. 5 we subsume approaches for programming collectives and analyze current implementations of swarm behavior for swarm robotic systems. In Sect. 6 we conclude our findings and point out possible future research challenges.

2 Challenges Resulting for Multipotent Systems

Extending our multipotent systems reference architecture [9] with virtual swarm capabilities for exploiting useful emergent effects and to easily program collective systems poses some challenges. In multipotent systems, robot ensembles being homogeneous at design time can become heterogeneous concerning their capabilities at run-time by combining physical reconfiguration on the hardware level with self-awareness. We aim at exploiting this property for enabling robots to implement the reference architecture to also adapt at run-time for participating in swarm algorithms. While we already provide the possibility of extending the range of domain-specific capabilities in multipotent systems when it is necessary, we want to reduce the effort a system designer needs to invest when integrating virtual capabilities. In our multipotent systems reference architecture (cf. Fig. 1), we integrate capabilities within the *semantic hardware layer* which is an interface to self-descriptive hardware [25]. The semantic hardware layer recognizes new hardware connected to the robot and updates the available capabilities respectively in a self-aware manner. It provides these capabilities to its superordinate *agent layer* that can make use of them when involved in an ensemble (coordinated on *ensemble layer*) that currently executes a task introduced on *task layer*.

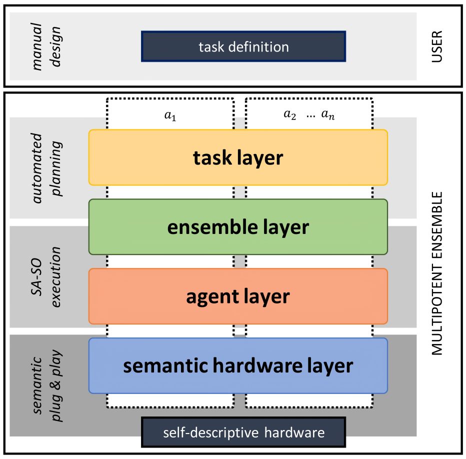

Fig. 1. The multipotent systems reference architecture for multi-robot ensembles, simplified version adapted from [14].

We generate tasks by automated planning on the task definition the system's user introduces through an interface on *task layer*. Agents $\alpha_{1..n} \in \mathcal{A}$ in the multipotent ensemble then allocate these tasks cooperatively to agents capable of solving the task. These agents then form an ensemble coordinated by one specific agent through its ensemble layer, e.g., α_1 (cf. Fig. 1). The ensemble then executes the respective task by the appropriate interplay of the coordinator's ensemble layer and the other ensemble members' agent layer. To enable the system to make use of such new capabilities that are coupled with physical hardware, an expert first needs to make changes to this core element of the system. Necessary adaptions include, e.g., extending the domain model of the ensemble appropriately, implementing the hardware access (drivers) accordingly, or integrating the new hardware physically into the system (hardware adapters, wiring). While adaptations of the domain model are necessarily required when a user introduces new hardware modules that offer new capabilities, e.g., an GAS$_x$ sensor module offering the previously unknown capability of *measuring*-GAS$_x$, we aim at avoiding this for virtual capabilities. If a capability is not directly associated with and not only available through the presence of dedicated physical hardware, e.g., for participating in swarm algorithm A instead of algorithm B or for executing a Protelis program C instead of program D, we aim at avoiding such modifications to the core system for certain classes of capabilities. Our challenge here is to identify such classes where it is possible to separate a fixed part from a variable part. Then, we can implement that fixed part into the system *once* at design-time as a *virtual capability*, and integrate the variable part dynamically at run-time as the virtual capability's *parameters*. Further, we also require to adapt our current mechanism for task execution accordingly. For realizing virtual capabilities aiming at collective behavior, we need to introduce the possibility of direct communication between instances of the same type of virtual capabilities which was only possible through agent layer up to now. Without the direct exchange of relevant information between participating entities many external programming approaches for ensembles can not function because they rely on some form of directly accessible messaging interfaces [19,20].

3 Approach

To be available to an agent $\alpha \in \mathcal{A}$ in the multipotent systems reference architecture [14], a capability requires a set of physical hardware modules, i.e., sensors and/or actuators (*S/A*). While the set of S/A does not need to be the same for every instantiation of a capability, we require the set of S/A to have the necessary user-specified functionality [5,25], e.g., determine the *presence* of an object. For their execution, capabilities do require a set of parameters, e.g., a direction vector for a *move* capability. In this paper, we refine this capability concept (cf. light-grey part of Fig. 2) by differentiating between virtual and physical capabilities (darker part of Fig. 2). Therefore, we demonstrate how we can combine already existing physical capabilities $\mathcal{C}^p$ for achieving collective behavior that we can parametrize in *virtual capabilities for collectives* $\mathcal{C}^v$. We apply this concept in a virtual capability for movement-vector based swarm behavior (cf. *Movement-Vector Based Swarm Capability* in Fig. 2) realizing the general pattern for individual agent participation in respective swarm algorithms. We further introduce a second virtual capability offering an interface between agents and their capabilities in our reference architecture and other collective programming approaches (cf. *External Collective Programming Capability* Fig. 2).

We assume that every agent can communicate with any other agent in the ensemble $\mathcal{E}$ it currently participates in. This is necessary to realize certain types of swarm behavior (e.g., particle swarm optimization PSO [27]) because we can not assume local sensors for all spatially distributed relevant values (e.g., measurements of other agents). Moreover, we can not assume to have perfect local sensors for every robot enabling it to externally determine the state of other robots precisely enough in a real-world setting. We achieve this by exploiting the communication middleware of the multi-agent framework Jadex [1]. With this framework, we can ease the conceptualization and implementation of our distributed multipotent systems through the use of Jadex Active Components, which are autonomously acting entities. Implementing each instance of $\mathcal{C}^p$, $\mathcal{C}^v$, and $\mathcal{A}$ as such active components and encapsulating their functionality in services each enables their direct interaction where this is necessary. We further assume that no outages (e.g., communication, sensor failures, broken robots) occur.

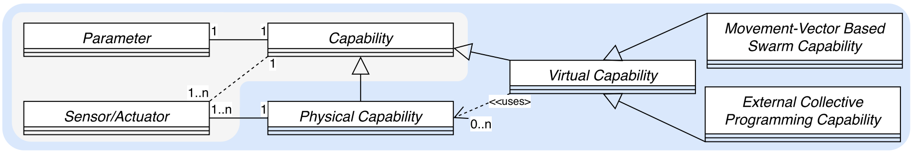

Fig. 2. General concept model for virtual capabilities. Instead of direct access to S/A, we provide access indirectly through associated physical capabilities.

3.1 Static and Dynamic Model of Virtual Capabilities

We differentiate between virtual capabilities $\mathcal{C}^v$ and physical capabilities $\mathcal{C}^p$ which both refine the previous concept of a capability, i.e., a service a robot provides for execution. In comparison to physical capabilities, virtual capabilities are not directly associated with S/A. Instead, for executing a virtual capability we require it to invoke associated other (physical) capabilities. Thus, virtual capabilities do only have indirect access to hardware but can be used to construct more complex behavior. Consequently, the set of parameters for a virtual capability needs to include additional information, e.g., the set of other capabilities it needs for its execution. This has also consequences for our currently established self-awareness [13], and self-organization mechanisms [12] we use to execute plans in multipotent systems. Because the execution of a virtual capability might require the cooperation within the ensemble $\mathcal{E} \subseteq \mathcal{A}$ executing it, we allow for every α_i executing a specific $c^v \in \mathcal{C}^v$ to directly exchange information with other $\alpha_{j \neq i}$ within the same ensemble that are executing the same instance of c^v. Further, communication is an urgent requirement for collective programming approaches we want to enable as external capabilities. We therefore separate each $c^v \in \mathcal{C}^v$ in an active part $c^{v:\text{ACT}}$ and a passive part $c^{v:\text{PAS}}$. While the active part differs for all $c^v \in \mathcal{C}^v$, we can define the passive part as a procedure $\text{RECEIVE}(c^v, \mathcal{V}_{\alpha_i})$ used for receiving relevant data $\mathcal{V}_{\alpha_i}$ from another agent α_i executing the same virtual capability c^v in general for all $c^v \in \mathcal{C}^v$. RECEIVE updates the values for these other agents stored in a shared map $\mathrm{M}^{\mathcal{E}} := \langle \alpha \in \mathcal{E} \rangle, \langle M^{\alpha} \rangle$ holding the most recent values M^{α} received from all $\alpha \in \mathcal{E}$. To enable the exchange of data, the active and passive part of each $c^v \in \mathcal{C}^v$ share this map. This means, when receiving $\mathcal{V}_{\alpha_{i \neq j}}$ in $c^{v:\text{PAS}}$, an agent α_j can update the entries referenced in $\mathcal{V}_{\alpha_i}$ concerning α_i in $\mathrm{M}^{\mathcal{E}}$ and subsequently access the data in $c^{v:\text{ACT}}$. In our code snippets, we indicate that α_j executing c^v sends $\mathcal{V}_{\alpha_j}$ to a specific other agent α_i executing the same instance of c^v with $\alpha_i.\text{SEND}(c^v, \mathcal{V}_{\alpha_j})$. Besides shared data and data received from other agents, in our algorithms we indicate local input with $\text{NAME} := \langle \text{INPUT}_1, ..., \text{INPUT}_n \rangle$.

3.2 Termination and Results of Virtual Capability Executions

Like for physical capabilities, we can define different termination types for virtual capabilities. Physical capabilities can terminate internally on their own or require external events for termination. A robot executing, e.g., its physical capability for moving to a certain position $c^p_{\text{MV\_POS}}$ can rely on the automatic

Algorithm 1. $c^{v:\text{FIN-COORD}} := \langle \text{F:AGGR}_{c^v}, \text{F:TERM}_{c^v} \rangle$

```
1: $\text{R}_{\text{AGGR}} \leftarrow \text{F:AGGR}_{c^v}(\text{M}^{\mathcal{E}})$    # aggregates the ensemble's current measurements
2: $\text{TERM} \leftarrow \text{F:TERM}_{c^v}(\text{R}_{\text{AGGR}})$    # decide for termination using the aggregated result
3: if TERM then
4:     $\text{STORE}(\text{R}_{\text{AGGR}})$    # if terminating, store the result for external evaluation
5:     for $\alpha_i \in \mathcal{E}$ do
6:         $\alpha_i.\text{SEND}(c^v, \text{TERM})$    # broadcast the termination decision in the ensemble
```

termination of $c^p_{\text{MV\_POS}}$ when it reaches the position defined in the parameters. Instead, a physical capability $c^p_{\text{MV\_VEC}}$ that moves a robot in a direction using a speed vector does not terminate itself as the movement does not have a natural end and thus needs to be terminated externally. Likewise, virtual capabilities can terminate their execution internally or require external termination. This is especially relevant for all virtual capabilities that implement collective behavior. We can define termination criteria with appropriate parameters for some swarm behavior, e.g., executing a virtual capability implementing a PSO can terminate itself when all agents in the swarm gather within a certain distance [27]. For other swarm behavior, e.g., achieving the equal distribution of robots in a given area with the triangle algorithm [15], we do not want to define such criteria (e.g., for achieving the continuous surveillance of that area) or even can not do it at all (e.g., for steering a swarm in one direction with guided flocking [2]) and thus rely on an external event for termination. Besides defining when to terminate a c^v implementing swarm behavior or other collective behavior, we also require to quantify the emergent effect of executing c^v and store it for up-following evaluation like we do with the results originating from physical capability executions. For PSO, e.g., we finally want to determine the position the highest concentration of a parameter an ensemble was searching for was measured. In this case, we can calculate the position of relevance by calculating the ensemble's center of gravity when the geometrical diameter of the swarm, i.e., the euclidean distance between the $\alpha_i, \alpha_j \in \mathcal{E}$ having the greatest distance between each other, gets lower than a user-defined threshold. For such calculations and to determine termination for virtual capabilities therewith, we extend the role of the ensemble coordinator that is responsible to coordinate a plan's execution [10]. Concerning the results of (physical) capability executions, the coordinator only acts as a pass-through station for results originating from any capability execution in the ensemble. The coordinator stores each result in a distributed storage and evaluates data when necessary, e.g., for deciding on the current plan's progress or during replanning on the task layer (cf. Fig. 1). To determine the termination of a virtual capabilities execution, we now enable the coordinator to also aggregate, analyze and post-process the intermediate results from virtual capabilities before storing them by using capability specific procedure $c^{v:\text{FIN-COORD}}$ (cf. Algorithm 1). Because we guarantee with an additional constraint in our constraint-based task allocation mechanism [6] that the agent adopting the coordinator role always also participates in the execution of the collective behavior, i.e., executes the

Algorithm 2. $c^{v:\text{ACT}}_{\text{SW}} := \langle C^p_{\text{SW}}, \text{CALC}_{\text{SW}}, \mathcal{E}_{\text{SW}} \rangle$

```
1: repeat
2:     for each $c_i \in C^p_{\text{SW}}$ parallel do
3:         $\text{M}^{\text{SELF}}[c_i] \leftarrow \text{EXEC}(c_i)$   # execute all relevant capabilities and store the results
4:     $\text{M}^{\mathcal{E}}[\text{SELF}] \leftarrow \text{M}^{\text{SELF}}$   # store local results in the map for all ensemble results
5:     for each $\alpha_i \in \mathcal{E}_{\text{SW}}$ parallel do
6:         $\alpha_i.\text{SEND}(c^v_{\text{SW}}, \text{M}^{\text{SELF}})$   # distribute stored results in the ensemble
7:     $\text{PAR}_{c^p_{\text{MV\_VEC}}} \leftarrow \text{CALC}_{\text{SW}}(\text{M}^{\mathcal{E}})$   # calculate the new movement vector
8:     $\text{EXEC}(c^p_{\text{MV\_VEC}})$   # update the current movement vector
9: until TERM   # decide on termination using the received value
```

respective $c^{v:\text{ACT}}$, it can also receive values other ensemble members send and thus has access to $\text{M}^{\mathcal{E}}$. By using an aggregation function F:AGGR_{c^v} taking $\text{M}^{\mathcal{E}}$ as input parameter that is specific for each c^v, we can quantify the emergent effect every time the entries in $\text{M}^{\mathcal{E}}$ change (L. 1 in Algorithm 1). If the termination criteria (F:TERM_{c^v} in Algorithm 1) holds for the current result (L. 2 in Algorithm 1), the coordinator can store that result in the distributed storage (L. 4 in Algorithm 1) and distribute the current termination state TERM within the ensemble (L. 6 in Algorithm 1). Each agent can receive this signal with a respective service $c^{v:\text{FIN-PART}}$ to receive the coordinator's termination signal TERM with $\text{RECEIVE}(c^v, \text{TERM})$. The service $c^{v:\text{FIN-PART}}$ shares TERM with the active part $c^{v:\text{ACT}}$ of c^v in $\text{TERM}^{\mathcal{E}}$, which we use to stop the execution of c^v. For $c^v \in \mathcal{C}^v$ that can terminate externally only, we can thus enable the user to also have the possibility to terminate the execution of c^v.

3.3 A Capability for Movement-Vector Based Swarm Algorithms

For achieving emergent effects generated by movement-vector based swarm behavior, we introduce a *Movement-Vector Based Swarm Capability* c^v_{SW} with its according parameters $\text{PAR}_{c^v_{\text{SW}}}$ (cf. Fig. 2). This virtual capability realizes swarm behavior from the class of movement-vector-based swarm algorithms such as the PSO [7,27], flocking [21], or the triangle formation [18] among others, that can be of use for multipotent systems. We illustrate the respective active part $c^{v:\text{ACT}}_{\text{SW}}$ of c^v_{SW} in Algorithm 2 that executes a general pattern capable of producing the mentioned swarm behaviors. In a first step, each agent executing c^v_{SW} measures and remembers relevant values according to the set of physical parameters C^p_{SW} included in $\text{PAR}_{c^v_{\text{SW}}}$ in parallel (cf. L. 3 in Algorithm 2). After finishing the execution of all capabilities in case of self-terminating capabilities or after starting to execute non-self-terminating capabilities respectively, agents executing c^v_{SW} in parallel exchange these local measurements M^{SELF} with all agents in the current ensemble $\mathcal{E}_{\text{SW}}$ that execute the same instance of c^v_{SW} (cf. L. 6 in Algorithm 2). Each agent $\alpha \in \mathcal{E}_{\text{SW}}$ remembers these measurements in the virtual capability's locally shared map $\text{M}^{\mathcal{E}}$ that holds the most recent values for all neighbors including itself (cf. L. 4 in Algorithm 2). By using this aggregated measurements $\text{M}^{\mathcal{E}}$, each agent then is able to determine the necessary adaption to its current move-

Algorithm 3. $c^v_{\text{EXT}} := \langle \text{PROG}_{\text{EXT}}, \text{PC}_{\text{EXT}}, \mathcal{E}_{\text{EXT}} \rangle$

```
1: repeat
2:     $\text{M}^{\mathcal{E}}_{\text{SNAP}} \leftarrow \text{M}^{\mathcal{E}}$    # create a snapshot of the current ensemble values
3:     $\langle C^p_{\text{EXT}}, \text{TERM}_{\text{EXT}}, \text{PC}_{\text{EXT}}, \mathcal{V}_{\text{EXT}} \rangle \leftarrow \text{PROG}(\text{PC}_{\text{EXT}}, \text{M}^{\mathcal{E}}_{\text{SNAP}})$    #execute the program
4:     for each $c_i \in C^p_{\text{EXT}}$ parallel do
5:         $\text{M}^{\text{SELF}}[c_i] \leftarrow \text{EXEC}(c_i)$    #execute capabilities required by the program
6:     $\text{M}^{\mathcal{E}}[\text{SELF}] \leftarrow \text{M}^{\text{SELF}}$    #store results for next iteration of the program
7:     for each $\alpha_i \in \mathcal{E}_{\text{EXT}}$ parallel do
8:         $\alpha_i.\text{SEND}(c^v_{\text{EXT}}, \mathcal{V}_{\text{EXT}})$    #distribute relevant data of the program
9: until $\text{TERM}_{\text{EXT}} \vee \text{TERM}$    #check termination set by the program or coordinator
```

ment vector (cf. L. 8 in Algorithm 2) for achieving the intended specific swarm behavior encapsulated in CALC_{SW} (cf. L. 7 in Algorithm 2). As all agents in $\mathcal{E}_{\text{SW}}$ repeatedly execute this behavior until a specific termination criteria TERM holds (passed over to the passive part $c^{v:\text{PAS}}$ of c^v_{SW} from the coordinator, cf. Sect. 3.2), they achieve the specific swarm algorithm's emergent effect collectively (cf. L. 9 in Algorithm 2). By adjusting CALC_{SW} in particular, we can exploit this generally implemented form of a virtual capability to execute different swarm algorithms that would require an individual implementation each otherwise.

3.4 An Interface for External Collective Programming Languages

During the design of multipotent systems, we can not foresee all necessary functionality in specific use cases a user of the system might have in mind. Therefore, we offer the possibility of external programming to the system's user. We do this by introducing virtual capabilities $c^v_{\text{EXT}} \in \mathcal{C}^v$ for external collective programming approaches which become a fixed part of the multipotent system and represent an interface to the run-time environment of a specific programming language each. In contrast to c^v_{SW}, where we need to define the actual calculation CALC_{SW} within the host system and its respective programming language (i.e., that the multipotent system reference architecture from Sect. 2 is implemented with), we are not restricted to that when using a specific c^v_{EXT}. Instead, we encapsulate necessary information in a program written in the respective external programming language and only need to define the interface for the communication of that programming language's execution environment and the multipotent system's implementation. These external programs then define, how values we generate within the multipotent system are used and transformed into instructions for the multipotent system. Like for any $c^v \in \mathcal{C}^v$, we enable each c^v_{EXT} to execute other already existing capabilities $c^p \in \mathcal{C}^p$ of the multipotent system, i.e., choose respective parameters and read results from those capabilities' execution that we store in $M^{\mathcal{E}}$ through the defined interface (L. 5 in Algorithm 3). This way, a user can program new complex behavior PROG_{EXT} in the external programming language while also using already available functionality provided by $\mathcal{C}^p$ within our system. The programmer only needs to know the interface to relevant $c^p \in \mathcal{C}^p$ and does not require further knowledge of the underlying multipotent system, e.g., if the PROG_{EXT} requires the change the current movement vector of the executing robot. For its execution, the respective c^{v}_{EXT} then uses $\textsc{prog}_{\text{EXT}}$ as an additional parameter (cf. Algorithm 3). This way, and to allow for changing the behavior of c^{v}_{EXT}, the programmer can dynamically exchange the external program at runtime. With the start of the capability execution within the active part of each $c^{v:\text{ACT}}_{\text{EXT}}$, we run $\textsc{prog}_{\text{EXT}}$ from its entry point by handing over a program pointer $\textsc{pc}_{\text{EXT}}$ and a snapshot of the current state of $\mathrm{M}^{\mathcal{E}}$ (initially empty, L. 2 and 3 in Algorithm 3). When the execution of $\textsc{prog}_{\text{EXT}}$ stops, we require it to return a data vector $\langle C^{p}_{\text{EXT}}, \textsc{term}_{\text{EXT}}, \textsc{pc}_{\text{EXT}}, \mathcal{V}_{\text{EXT}}\rangle$ encapsulating instructions from the external program to the multipotent system. The first entry indicates whether the external program's control flow requires that physical capabilities C^{p}_{EXT} get executed in the following (L. 4 and 5 in Algorithm 3). The second entry determines, whether $\textsc{prog}_{\text{EXT}}$ already reached its termination criteria $\textsc{term}_{\text{EXT}}$ and the execution of c^{v}_{EXT} can be finished internally (L. 9 in Algorithm 3). The third entry determines, what the next program counter $\textsc{pc}_{\text{EXT}}$ is if $\textsc{term}_{\text{EXT}}$ does not hold. Because information on which values need to be within the ensemble $\mathcal{E}_{\text{EXT}}$ is encapsulated in $\textsc{prog}_{\text{EXT}}$ but the distribution itself is performed by the multipotent system's agent communication interface, in a fourth entry $\mathcal{V}_{\text{EXT}}$ determines those values (L. 7 and 8 in Algorithm 3). While $\textsc{term}_{\text{EXT}}$ does not hold and no termination signal is received from the coordinator of $\mathcal{E}_{\text{EXT}}$ in $c^{v:\text{FIN-PART}}$ (cf. Sect. 3.2), the execution of c^{v}_{EXT} continues to execute $\textsc{prog}_{\text{EXT}}$ with the current $\textsc{pc}_{\text{EXT}}$ in the following iteration. Thereby, it uses an updated version of $\mathrm{M}^{\mathcal{E}}$ (L. 2 in Algorithm 3) containing latest local values (L. 6 in Algorithm 3) as well as such received in $c^{v:pas}_{\text{EXT}}$ meanwhile (Sect. 3.1). Each $\textsc{prog}_{\text{EXT}}$ adhering to this convention thus can access the set of locally available physical capabilities and use the communication middleware of our multipotent system in the current ensemble. This creates a high degree of flexibility in the way of programming with our approach.

4 Proof of Concepts

To demonstrate the flexibility of our approach we give proof of concepts in the following. We, therefore, implemented a virtual capability for movement-vector based swarm algorithms c^{v}_{SW} and evaluated it with different parameters to achieve different emergent effects. We demonstrate the concept of a virtual capability for the movement-vector based swarm behavior with video materials[1] isolated in a NetLogo simulation[2] and integrated with our multipotent systems reference implementation. Further, we demonstrate the feasibility of integrating an external programming language for collectives as a virtual capability by example.

4.1 Executing Movement-Vector Based Swarm Algorithms

We validate the concept of the virtual capability for movement-vector based swarm algorithms c^{v}_{SW} we introduced in Sect. 3.3 using different parameters for

[1] https://github.com/isse-augsburg/isola2020-swarm-capabilities.git.
[2] NetLogo download on https://ccl.northwestern.edu/netlogo/download.shtml.

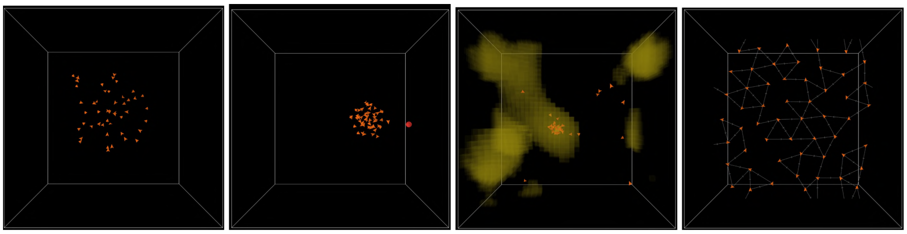
(a) Gathering (b) Guided Boiding (c) PSO (d) Triangle

Fig. 3. Screen shots of a simulation environment showing the use of a swarm capability for different parameters resulting in respective emergent effects (top down perspective). See footnotes 1, 2 for video material and a respective NetLogo simulation source file.

realizing different emergent effects. In a simplified major catastrophe scenario, a firefighter might want to a) gather its ensemble of mobile robots, b) move them collectively to the area where, e.g., a gas accident happened, c) search for the source of the gas leak, and d) survey the area close to the leak (video materials on our GitHub). We can instruct our system, e.g., with our task-orchestration approach for ensembles Maple-Swarm [11]. To handle this scenario we can use the c^v_{SW} with different sets of parameters in steps a)-d) each (cf. Figs. 3a to 3d), illustrating the flexibility of our concept of c^v_{SW} including its termination functionality. For all instances of c^v_{SW} we execute to realizing the desired emergent effect for achieving a)-d), we assume the following: A sufficiently equipped ensemble $\mathcal{E}_{\text{SW}}$ is available concerning the set of physical capabilities C^p_{SW} necessary for that concrete instantiation which we can achieve, e.g., with our self-aware and market-based task allocation mechanism [13] in combination with our self-organized resource allocation mechanism [6]. For each result of CALC, we normalize (NORM) the resulting distance (DIST) vector originating from the robots current position POS_α and scale it with the robots maximum velocity with ν. We assume a working collision avoidance system provided by the robotics controller.

a) For *gathering* the ensemble, we can execute c^v_{SW} with $C^p_{\text{SW}} := \{c^p_{\text{POS}}\}$, where c^p_{POS} measures the executing robot's current position (cf. Fig. 3a). Each robot can terminate the execution of c^v_{SW} locally when the diameter $\text{DIAM}()$ of the swarm is below a user-defined threshold x, calculated with the measurements available in $\text{M}^{\mathcal{E}}$, i.e., $\text{TERM}_{\text{SW}} := \text{DIAM}(\text{M}^{\mathcal{E}}[*][c^p_{\text{POS}}]) \leq x$. We calculate the desired moving vector using the ensemble's center of gravity $\text{GRAV}()$, i.e., $\text{CALC}() := \nu \cdot \text{NORM}(\text{DIST}(\text{POS}_\alpha, \text{GRAV}(\text{M}^{\mathcal{E}}[*][c^p_{\text{POS}}])))$. Both, $\text{DIAM}()$ and $\text{GRAV}()$ only require information concerning the position of each robot in $\mathcal{E}_{\text{SW}}$, thus results from executing c^p_{POS} stored in $\text{M}^{\mathcal{E}}$ are sufficient therefore.

b) For *controlling the ensemble to a goal location* with an adapted flocking approach following the idea of boiding in [21], we execute c^v_{SW} with $C^p_{\text{SW}} := \{c^p_{\text{POS}}, c^p_{\text{VEL}}\}$, where c^p_{POS} measures the executing robot's current position and c^p_{VEL} its current velocity (cf. Fig. 3b). We can calculate the desired moving vector by appropriately weighting the three urges for the cohesion COH of the ensemble, the separation $\textsc{sep}$ from the closest neighbor in the ensembles, and the alignment $\textsc{ali}$ of the individual robot's moving direction with that of the ensemble known from [21]: $\textsc{calc} := \omega_1 \cdot \textsc{sep}(\mathrm{M}^{\mathcal{E}}[*][c^p_{\textsc{pos}}]) + \omega_2 \cdot \textsc{coh}(\mathrm{M}^{\mathcal{E}}[*][c^p_{\textsc{pos}}]) + \omega_3 \cdot \textsc{ali}(\mathrm{M}^{\mathcal{E}}[*][c^p_{\textsc{vel}}])$. To guide the ensemble to the goal location we exploit how ensemble members evaluate $\mathrm{M}^{\mathcal{E}}$ for adapting their movement vector (L. 7 in Algorithm 2) by adding an additional entry for a non-ensemble member (i.e., a dedicated leader robot or any other position-aware device) that also measurements of C^p_{SW} frequently. Because all ensemble members use the complete map $\mathrm{M}^{\mathcal{E}}$, the emergent effect is what we aim for: guiding the collective to a goal location the non-ensemble robot is moving to. Robots can not terminate the execution of c^v_{SW} locally in this case because they have no information on the goal location and thus rely on an external termination signal $\textsc{term}$ from their coordinator (who possibly requires to receive it from the user itself).

c) For *searching for the highest concentration* of a certain parameter, we execute c^v_{SW} with an adapted version of the particle swarm optimization algorithm (PSO) [27] (cf. Fig. 3c). Obviously, we require to contain the respective capability for measuring the parameter of interest $c^p_{\textsc{par}}$ in C^p_{SW}, in addition to $c^p_{\textsc{pos}}$ and $c^p_{\textsc{vel}}$, i.e., $C^p_{\mathrm{SW}} := \{c^p_{\textsc{par}}, c^p_{\textsc{pos}}, c^p_{\textsc{vel}}\}$. To determine the movement vector of robot α, we define $\textsc{calc} := \omega_1 \cdot \textsc{dist}(\textsc{pos}_\alpha, \textsc{max}(\textsc{max}(\mathrm{M}^{\mathcal{E}}[\textsc{self}][c^p_{\textsc{par}}], \textsc{max}^{\textsc{self}}))) + \omega_2 \cdot \textsc{dist}(\textsc{pos}_\alpha, \textsc{max}(\textsc{max}(\mathrm{M}^{\mathcal{E}}[*][c^p_{\textsc{par}}], \textsc{max}^{\mathcal{E}})) + \omega_3 \cdot \textsc{dist}(\textsc{pos}_\alpha, \textsc{rand}(x, y, z))$ as the weighted sum of distance vectors pointing from the robot α's current position $\alpha_{\textsc{pos}}$ to the position with the iteratively updated highest measurement of the parameter of interest from the robot itself $\textsc{max}^{\textsc{self}}$, the whole ensemble $\textsc{max}^{\mathcal{E}}$, and a random direction $\textsc{rand}(x, y, z)$ included for exploration. Similar to the execution of c^v_{SW} for gathering in a), we can let the agents in the ensemble decide on the termination on c^v_{SW} by determining whether the diameter of the ensemble is below a threshold x, i.e., $\textsc{term}_{\mathrm{SW}} := \textsc{diam}(\mathrm{M}^{\mathcal{E}}[*][c^p_{\textsc{pos}}]) \leq x$.

d) For realizing the *distributed surveillance* of an area of interests, we adapted the triangle formation algorithm from [15] to also work within a 3D-environment (cf. Fig. 3d). With this algorithm, we can exploit the emergent effect of a swarm distributing in an area holding a predefined distance s to each other at a given height h. To produce the desired emergent effect, a robot α requires position measurements of its two closest neighbors only, i.e., $C^p_{\mathrm{SW}} := \{c^p_{\textsc{pos}}\}$. To determine the required movement vector, we first need to determine the two closest neighbors $\alpha_{1,2}$ of α in the ensemble, i.e., $\neg\exists\alpha_i \in \mathcal{E} : \textsc{dist}(\alpha, \alpha_i) < \textsc{dist}(\alpha, \alpha_1) \wedge \neg\exists\alpha_i \in \mathcal{E} \setminus \alpha_1 : \textsc{dist}(\alpha, \alpha_i) < \textsc{dist}(\alpha, \alpha_2)$. We then calculate the center of gravity $\textsc{grav}(\alpha_1, \alpha_2)$ between α_1 and α_2 and determine the distance vector pointing from α to the closest intersection point of the plane at height h (defined parallel to ground level) and the circle around the center of gravity with radius $\sqrt{3} \cdot \frac{s}{2}$ (being perpendicular to the straight defined by α_1 and α_2) as the goal position of α. While we can define a condition for termination of the execution of c^v_{SW}, e.g., in case that all distances between closest neighbors only vary marginally for all robots in the ensemble, we do not want to specify such in the case of continuous surveillance. Like in b), we require an external termination $\textsc{term}$ signal from the user or another external entity.

```
1 module count_neighbors
2 let num_of_neighbors = sumHood(nbr(1))
3 num_of_neighbors
```

```
1 module term_after_iterations
2 def iterations () = rep(x <- 0) { x + 1 }
3 def term_after(x) =
4       if ( iterations () > x) { self .term() }
5       else { iterations () }
6 terminate_after(10)
```

```
 1 module measure_temp
 2 import ParamFactory.get;
 3 def measure_temp() {
 4   let cap_type = self.getType("temp")
 5   let measurement_param = get(cap_type)
 6   let param = measurement_param.get()
 7   param.set("measureOnce", true)
 8   let temp = self.request(param, cap_type)
 9   temp
10 }
11 measure_temp()
```

Fig. 4. Minimal Protelis programs demonstrating the feasibility of the integration: Communication between agents (top left), enforcing the self-termination from the host system (bottom right), and accessing to capabilities of the host system (right).

4.2 Protelis as an Example for an External Virtual Capability

We demonstrate the feasibility of integrating an external programming language into the multipotent systems reference architecture by example. Therefore, we instantiate the concept of an external collective programming capability with c^v_{PROT} providing an interface for the Protelis Aggregate Programming approach [19]. To validate the concepts we introduced in Sect. 3.4, we give a proof of concepts concerning the relevant parts executing an external capability. These concepts are the *communication* between participating agents, commanding the *execution* and making use of the *results* of capabilities running on the host system, and ensuring *self-termination* of the external capability, if necessary. According to [19], for communication between entities, Protelis requires a network manager. With c^v_{PROT} we implement such (L. 7 in Algorithm 3). We can validate its functionality with the minimal example of a Protelis program we give in Fig. 4 (top left) that counts all members of the ensemble using the nbr construct in L. 2 in Fig. 4 (top left). The example showcases the ability of communication between agents executing c^v_{PROT}. In the Protelis program in Fig. 4 (right), we demonstrate how external capabilities can define required access to physical capabilities of the multipotent system host system (implemented in JAVA) using the self construct of Protelis for measuring temperature (L. 11 in Fig. 4 - right). In L. 4–7 of Fig. 4 (right), we access the knowledge base of our architecture by importing the ParamFactory (L. 2 in Fig. 4 - right). We use this knowledge base for loading the correct format of the necessary parameters for the measure temperature capability. For achieving this, we make use of the JAVA Reflection API. With self.request (L. 8 in Fig. 4 - right), we define the request the external capability has concerning the execution of physical capabilities (L. 3 in Algorithm 3) whose result we return in L. 9 in Fig. 4 (right) when it is available. To avoid the blocking of the Protelis program's execution when it requests a capability execution, we implement the data interface to our multipotent system as a reload cache. To validate the correct program flow and validate correct self-termination of c^v_{PROT}, in the Protelis program we give in Fig. 4 (top left) we let each member of the ensemble iterate a counter (L. 6 in Fig. 4 - bottom left). Because there is no access to physical capabilities included in the program, the execution of each instance terminates after 10 iterations and accordingly notifies the encapsulating external capability c^v_{PROT} with TERM_{EXT} evaluating true when it finally reaches self.terminate() in L. 4 in Fig. 4 (bottom left). Thus, we demonstrate the feasibility of integrating an interface between Protelis and our multipotent systems reference architecture with a specific virtual capability as a proof of concepts for our concept of from Sect. 3.4. We provide video material for demonstration purposes on GitHub. The integration of c^v_{PROT} currently is limited to only execute one Protelis program per agent in parallel and relies on capabilities provided by the host system to terminate on their own (cf. Sect. 3.2).

5 Related Work

The literature on swarm behavior, swarm algorithms, or swarm intelligence is manifold. When swarm behavior should be exploited in a real-world application, there are two common directions researchers currently follow. The first direction is that of focusing on one specific behavior found in nature that gets analyzed and migrated to technical systems. Examples for that direction are manifold, thus we only can give an excerpt of research relevant for this paper. To achieve a collective transport of an object, the authors in [4, 17] developed a specialized controller by using an evolutionary algorithm for mobile ground robots. While they achieve the desired effect, suffer from the evolutionary algorithms inherent properties of high specialization and the lack of generality: The generated controller can not be used in any other use case. To achieve a close-to equal distribution of swarm entities in a given area, e.g., for distributed surveillance, the authors in [16] adapt a potential-field based deployment algorithm. Unfortunately, the algorithm thus can only be used for exactly that use case. While the authors of [15] propose that they can adapt their swarm approach for distributed surveillance to also achieve flocking and obstacle avoidance they, unfortunately, do not further investigate in this direction. In our opinion, this is a step in the right direction to generate a general pattern for achieving swarm behavior which we try to make with our approach. In [23] the authors adapt the particle swarm optimization algorithm (PSO) [27] for the use of UAV in disaster scenarios to explore an area and detect victims. While the authors can adapt parameters to achieve different goals, the approach is still limited to that narrowly defined area and can not easily be extended. With an adapted flocking algorithm based on the approach of [21], the authors in [24] demonstrate how UAVs can achieve swarm behavior that is very close to that of natural swarms. Unfortunately, the implementation is very specific and can solely achieve this specific swarm behavior.

The second direction researchers follow is that of abstracting from specific applications and use cases and developing a general framework for collective behavior that can be programmed or parametrized in different ways. There already exist interesting approaches for programming collective behavior addressed in the ASCENS project [26]. Protelis [19] is one approach we also categorize in this direction. The authors center it around the idea of abstracting entities in a collective system as a point in a high dimensional vector field. Programming of the collective happens by performing operations on that field. By using implicit communication between entities, the programmer can achieve that changes performed in these fields are distributed within the collective. While a user can exploit this behavior to implement complex collective on an abstract level, it is not easy to achieve swarm behavior for complex mobile robot tasks solely with Protelis. Its lack of general hardware integration and a general task concept necessary for goal-oriented robot collaboration requires Protelis to be integrated into a further framework as we perform it in this paper. Another programming language aiming at collective systems is Buzz [20]. In comparison to Protelis, the authors of Buzz directly aim at integrating their programming language within robot operating systems. They provide swarm primitives for achieving a certain desired collective behavior each. Unfortunately, Buzz also lacks a concept for goal-oriented task orchestration. Further and like for using Protelis, a user of Buzz currently requires a system specifically designed for the respective programming language. With our approach, we can overcome this by providing the possibility to use programs written with any of the two languages in an integrated task orchestration framework. Further, we also try to find some general abstraction from specific applications and use cases in our approach. Moreover, we can use it to analyze and implement specific swarm behavior. Thus, we try to close the gap between the two methods currently existing in the literature.

6 Conclusion

The research community already exploits the positive properties of swarm behavior like robustness and scalability within many different approaches for controlling the behavior of collective adaptive systems. In this paper, we demonstrated how we can subsume many of these approaches by extracting their general swarm behavior in a virtual capability for movement-vector based swarm algorithms. We integrated this virtual capability into our reference architecture for multipotent systems. We further demonstrate how we can use instances of virtual capabilities to provide adapters to other programming approaches for collective systems on the example of Protelis [19]. Thus, virtual capabilities, in general, can compose existent capabilities of robots, i.e., complexly integrate already provided robot services, which we can exploit to create collective behavior in ensembles. In future work, we will elaborate on if and how we can drop our current assumption of having a steady communication link between ensemble members. This will help us to better deal with failures or complete break down of robots.

References

1. Braubach, L., Pokahr, A.: Developing distributed systems with active components and jadex. Scalable Comput. Pract. Experience **13**(2), 100–120 (2012)

2. Celikkanat, H., Turgut, A.E., Sahin, E.: Guiding a robot flock via informed robots. In: Asama, H., Kurokawa, H., Ota, J., Sekiyama, K. (eds.) Distributed Autonomous Robotic Systems, pp. 215–225. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00644-9_19
3. Dedousis, D., Kalogeraki, V.: A framework for programming a swarm of UAVs. In: Proceedings of the 11th Pervasive Technologies Related to Assistive Environment Conference, pp. 5–12 (2018)
4. Dorigo, M., et al.: The SWARM-BOTS project. In: Şahin, E., Spears, W.M. (eds.) SR 2004. LNCS, vol. 3342, pp. 31–44. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30552-1_4
5. Eymüller, C., Wanninger, C., Hoffmann, A., Reif, W.: Semantic plug and play - self-descriptive modular hardware for robotic applications. Int. J. Semant. Comput. (IJSC) **12**(04), 559–577 (2018)
6. Hanke, J., Kosak, O., Schiendorfer, A., Reif, W.: Self-organized resource allocation for reconfigurable robot ensembles. In: 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pp. 110–119 (2018)
7. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE (1995)
8. Kosak, O.: Facilitating planning by using self-organization. In: 2017 IEEE 2nd International Workshops on Foundations and Applictions of Self* Systems (FAS*W), pp. 371–374 (2017)
9. Kosak, O.: Multipotent systems: a new paradigm for multi-robot applications. In: Organic Computing: Doctoral Dissertation Colloquium, vol. 10, p. 53. kassel University Press GmbH (2018)
10. Kosak, O., Bohn, F., Keller, F., Ponsar, H., Reif, W.: Ensemble programming for multipotent systems. In: 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 104–109 (2019)
11. Kosak, O., Huhn, L., Bohn, F., et al.: Maple-swarm: programming collective behavior for ensembles by extending HTN-planning. In: 9th International Symposium on Leveraging Application of Formal Methods, Verification and Validation (2020)
12. Kosak, O., Wanninger, C., Angerer, A., et al.: Decentralized coordination of heterogeneous ensembles using jadex. In: IEEE 1st International Workshops on Foundations and Application of Self* Systems (FAS*W), pp. 271–272 (2016)
13. Kosak, O., Wanninger, C., Angerer, A., et al.: Towards self-organizing swarms of reconfigurable self-aware robots. In: IEEE International Workshops on Foundations and Applications of Self* Systems, pp. 204–209. IEEE (2016)
14. Kosak, O., Wanninger, C., Hoffmann, A., Ponsar, H., Reif, W.: Multipotent systems: combining planning, self-organization, and reconfiguration in modular robot ensembles. Sensors **19**(1), 17 (2018)
15. Li, X., Ercan, M.F., Fung, Y.F.: A triangular formation strategy for collective behaviors of robot swarm. In: Gervasi, O., Taniar, D., Murgante, B., Laganà, A., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2009. LNCS, vol. 5592, pp. 897–911. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02454-2_70
16. Ma, M., Yang, Y.: Adaptive triangular deployment algorithm for unattended mobile sensor networks. IEEE Trans. Comput. **56**(7), 847–946 (2007)
17. Mondada, F., Gambardella, L.M., Floreano, D., et al.: The cooperation of swarm-bots: physical interactions in collective robotics. IEEE Rob. Autom. Mag. **12**(2), 21–28 (2005)

18. Nishimura, Y., Lee, G., Chong, N.: Adaptive lattice deployment of robot swarms based on local triangular interactions. In: 2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence, pp. 279–284 (2012)
19. Pianini, D., Viroli, M., Beal, J.: Protelis: practical aggregate programming. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1846–1853. ACM (2015)
20. Pinciroli, C., Beltrame, G.: Buzz: an extensible programming language for heterogeneous swarm robotics. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3794–3800 (2016)
21. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. ACM SIGGRAPH Comput. Graph. **21**(4), 25–34 (1987)
22. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014)
23. Sánchez-García, J., Reina, D., Toral, S.: A distributed PSO-based exploration algorithm for a UAV network assisting a disaster scenario. Fut. Gener. Comput. Syst. **90**, 129–148 (2019)
24. Vásárhelyi, G., Virágh, C., Somorjai, G., et al.: Outdoor flocking and formation flight with autonomous aerial robots. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3866–3873 (2014)
25. Wanninger, C., Eymüller, C., Hoffmann, A., Kosak, O., Reif, W.: Synthesizing capabilities for collective adaptive systems from self-descriptive hardware devices bridging the reality gap. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11246, pp. 94–108. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03424-5_7
26. Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.): Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16310-9
27. Zhang, Y., Wang, S., Ji, G.: A comprehensive survey on particle swarm optimization algorithm and its applications. Math. Prob. Eng. **2015** (2015)