Maple-Swarm: programming collective behavior for ensembles by extending HTN-planning

Oliver Kosak, Lukas Huhn, Felix Bohn, Constantin Wanninger, Alwin Hoffmann, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Kosak, Oliver, Lukas Huhn, Felix Bohn, Constantin Wanninger, Alwin Hoffmann, and Wolfgang Reif. 2020. "Maple-Swarm: programming collective behavior for ensembles by extending HTN-planning." *Lecture Notes in Computer Science* 12477: 507–24. https://doi.org/10.1007/978-3-030-61470-6_30.

Maple-Swarm: Programming Collective Behavior for Ensembles by Extending HTN-Planning

Oliver Kosak(✉), Lukas Huhn, Felix Bohn, 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. Programming goal-oriented behavior in collective adaptive systems is complex, requires high effort, and is failure-prone. If the system's user wants to deploy it in a real-world environment, hurdles get even higher: Programs urgently require to be situation-aware. With our framework Maple, we previously presented an approach for easing the act of programming such systems on the level of particular robot capabilities. In this paper, we extend our approach for ensemble programming with the possibility to address virtual swarm capabilities encapsulating collective behavior to whole groups of agents. By using the respective concepts in an extended version of hierarchical task networks and by adapting our self-organization mechanisms for executing plans resulting thereof, we can achieve that all agents, any agent, any other set of agents, or a swarm of agents execute (swarm) capabilities. Moreover, we extend the possibilities of expressing situation awareness during planning by introducing planning variables that can get modified at design-time or run-time as needed. We illustrate the possibilities with examples each. Further, we provide a graphical front-end offering the possibility to generate mission-specific problem domain descriptions for ensembles including a light-weight simulation for validating plans.

Keywords: Task orchestration · HTN-Planning · Swarm behavior · Robot swarms · Multi-agent systems · Multipotent systems

1 Motivation

The range of versatile applications for collective adaptive systems and especially for multi-robot systems steadily increased during the last years due to the potential benefits these applications can deliver for research, our daily life, or society in general. We can find examples that already profit from this development everywhere, e.g., for research in space exploration [18] or meteorological science [4,13,22], for autonomous search and rescue in major catastrophe scenarios [2,15], among many others. One crucial hurdle that every application needs to

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

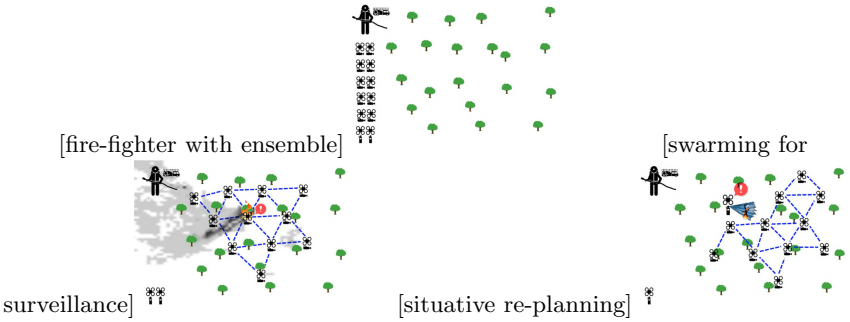


Fig. 1. Fire-fighter orchestrating an ensemble to deal with a forest fire scenario.

take before a user can actually profit from it is that of proper task-orchestration for the collective. Unfortunately, the current trend is that instead of aiming for a generic solution for that problem, every single new application requires a new software approach for its realization [3,9]. Besides varying hardware requirements [21], the often high complexity of performing specific goal-oriented task orchestration and planning for such ensembles hinders the reuse of previously successfully applied approaches for a broader set of applications. Achieving a general approach becomes even more complex as tasks show a high versatility or the user requires the ensemble to act in different problem domains. With our approach Maple [12], we already developed a task orchestration and execution framework for multipotent robot ensembles [15] having the potential to fill that gap. 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 [10].

In this paper, we demonstrate how we extend our approach to *Maple-Swarm* for supporting the operation of whole collectives by introducing the concepts of *agent groups* and virtual *swarm capabilities*. Swarm capabilities encapsulate collective behavior, where the local interaction of individuals provides useful emergent effects on the ensemble level, e.g., for distributing in an area with a potential field algorithm [20] or searching for the highest concentration of a parameter with an adapted particle swarm optimization algorithm [23]. We assume that we can alternate the specific swarm behavior with different parameters for the swarm capability like we present in [11]. To integrate these new concepts, we adapt to how we perform the designing, task planning, and task allocation process of Maple. In Maple-Swarm, we further extend the concept of hierarchical task networks (HTN) [6] we use for defining partial plans for specific situations and for generating situation-aware plans with automated planning at run-time. With agent groups, addressing *all agents*, *any agent*, *a set of agents*, or a *swarm of agents*, we extend the flexibility the multipotent system has for executing tasks. We thereby further increase the autonomy of multipotent robot ensembles that can choose which concrete agents adopt the respective roles at run-time in a self-organized manner while still preserving the possibility for the user to keep control over actions of particular robots, when necessary.

The running example we use for illustration purposes assumes a fire-fighter requiring to handle a forest fire scenario (cf. Fig. 1). Within a defined area, fires may ignite spontaneously. The fire-fighter needs to instruct its available ensemble of mobile robots to move to that area, continuously observe it, identify new fires, and extinguish them as fast as possible. Because of the size of the area, the high amount of robots available, and other urgent tasks only a human can accomplish, it is not always a feasible option for the fire-fighter to manually define routes for all robots or to react ad-hoc to newly identified fires. Instead, the fire-fighter wants to specify how the ensemble should react in different situations on an abstract level and let the system act appropriately and as autonomously as possible. Then, the system should decide on what to do according to its current state and that of the environment, e.g., by applying useful collective behavior.

The remainder of the paper is structured as follows. In Sect. 5 we subsume approaches for solving the problem of task orchestration for ensembles. In Sect. 2, we briefly reflect on the current state of Maple and its integration in our reference architecture for multipotent systems and illustrate our objectives. In Sect. 3, we propose our solution Maple-Swarm. In Sect. 4 we demonstrate the functionality of our approach for the firefighter scenario as proof of concepts. In Sect. 6 we conclude our findings and point out possible future research challenges.

2 Current State and Objectives

In multipotent systems, we generally differentiate between the *user device* and the *multipotent ensemble* consisting of multiple agents implementing a layered software architecture (cf. Fig. 2). The user device offers the possibility for designing problem domain definitions and thereby acts as an interface to the multipotent ensemble. The different layers each agent in the multipotent ensemble implements encapsulate their own functionality and communication pattern.

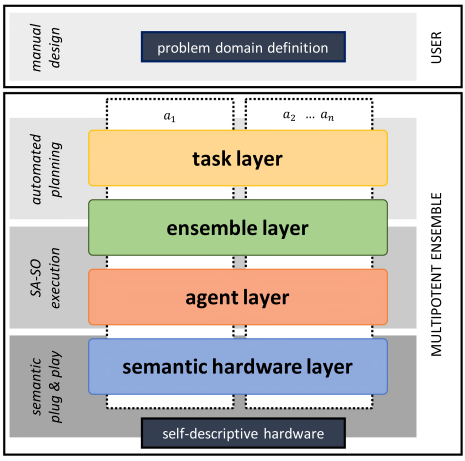


Fig. 2. Simplified multipotent systems reference architecture for ensembles from [10].

On the lowest layer, we enable hardware access to self-descriptive hardware with the *semantic hardware layer*. A self-awareness mechanism that detects changes in the respective agent's physical hardware configuration updates the set of capabilities the agent knows it can currently execute with that hardware autonomously [7]. On the superordinate *agent layer* and *ensemble layer*, we implement the necessary self-organization mechanisms to form ensembles with a market-based task allocation approach [14] and to autonomously execute tasks [13] introduced by the

task layer. On task layer, we evaluate the user-specified problem domain definition against the current state of the world the system is currently aware of and generate plans for this situation with an automated planner. We integrated this automated planner in our approach for a multi-agent script programming language for multipotent ensembles (Maple) [12]. There, we extend the approach of hierarchical task network (HTN) planning [5] for defining a problem domain and generating plans. We prefer this approach of plan-space planning over that of state-space planning [8] because of its efficiency and a higher level of control in real-world applications [6]. Plans in HTN are not "invented" by the planner (as in state-space planners), but selected and combined from a set of predefined partial plans. This achieves more control over the system where it is urgently needed, e.g., our fire-fighter scenario from Sect. 1. We use partial plans to define how robots need to execute capabilities for successfully accomplishing the plan, i.e., define partial plans on the level of the particular robot's capabilities. According to the situation of the system that is defined by the world state, the automated *planning* in Maple then can generate plans that are relevant for the respective situation. The multipotent system itself then updates the world state by executing plans and thereby generates new situations. To make plans executable, Maple includes a mechanism to *transform* generated plans into executable tasks that include necessary coordination information. We provide the possibility to define sequential, concurrent, alternative, and repeated as well as parallel and synchronized capability *execution*. For the concrete execution of plans, we let the multipotent system autonomously form appropriate ensembles at run-time.

Challenges for Maple-Swarm: For integrating agent groups and swarm capabilities encapsulating collective behavior in Maple-Swarm, we obviously need to adjust the way we *design problem domains* and *generate plans* for ensembles. While in Maple, we can already design tasks for the ensemble by using a common knowledge base enlisting all possible robot capabilities including their parameters and return values, we need to adapt this knowledge base accordingly for swarm capabilities. For the valid application of the application of swarms we present in the following, we assume swarms of robots to be ensembles of potentially heterogeneously configured robots that nevertheless are capable of executing the capabilities necessary for the respective swarm behavior (e.g., move with a certain movement vector, communicate measurements with each other). While we do not want to investigate the concrete execution of swarm capabilities here (we focus on this in [11]), we nevertheless require to define an appropriate interface including possible parameters and return types of swarm capabilities. This is necessary for initializing the task designer interface that requires a fully described capability knowledge base as well as for using return values of swarm capabilities during plan design and planning. We thus need to integrate the results of swarm capabilities in our concept of planning variables and adapt our planning mechanism accordingly. Further, we require to integrate the concepts for the respective agent groups we want to enable the problem domain designer to make use of in partial plans.

Addressing a group of robots with agent groups in a swarm capability requires an *adaptation of the market based, self-awareness enabled task-allocation mechanism* [13] we currently use in Maple for executing plans. To achieve this adaptation, we need to address two related challenges: Our task-allocation mechanism relies on the *self-awareness* information describing whether a robot can provide all required robot capabilities a task requires, delivered by the semantic hardware layer (cf. Fig. 2). Each robot uses this information it has available locally for validating its fitness for participating in that task. In case of a task requiring a swarm capability, we need to derive necessary self-awareness information on whether the robot can execute the task according to the capability's parametrization at run-time. A specific robot might be able to provide a swarm capability implementing a dedicated collective behavior for *some* parametrization but not for *all*: In case the swarm capability encapsulates the collective behavior of an adapted particle swarm optimization (PSO) [23] algorithm like we present in [10], the search parameter can require a wide range of different robot capabilities for performing measurements, depending on the concrete parameter specified by the user. The robot does not necessarily have all of these capabilities available and thus is not capable of participating in all possible instances of the swarm capability. Thus, we need to extend our self-awareness mechanism to not only provide information on whether a capability is *available* but also if it is *executable*, i.e., if the particular agent has all capabilities available that the swarm capability addresses with its specific parameters.

We further need to enable the *task-allocation mechanism* to deal with agent groups. In Maple [12], we can transform generated plans into tasks that address specific agents at any time (e.g., directly after planning). Now, we need to perform this transformation with respect to the current situation at run-time before we allocate them to actual robots. While in Maple, capabilities included in partial plans directly address particular agents, this is no longer the case in Maple-Swarm. Partial plans can contain any combination of capabilities, either addressing particular agents or any type of agent group. Thus, we require to adapt the requirements for tasks included in plans concerning the set of necessary capabilities a robot must provide for being able to work on that task accordingly. We need to do this appropriately for all possible combinations of capabilities addressing *particular agents*, *any agent*, *all agents*, a *set of agents*, or a *swarm of agents*. Also when a plan includes these agent groups, we need to determine the tasks we actually require and generate them at run-time.

Assumptions: We assume that in the simplified multipotent system, we evaluate our task orchestration framework with and perform task allocation in to validate the functionality of Maple-Swarm, we already have an appropriately configured system to abstract from physical reconfiguration needed otherwise. If necessary, we can create such situations, e.g., with our self-organized resource allocation mechanism we already proposed in [7].

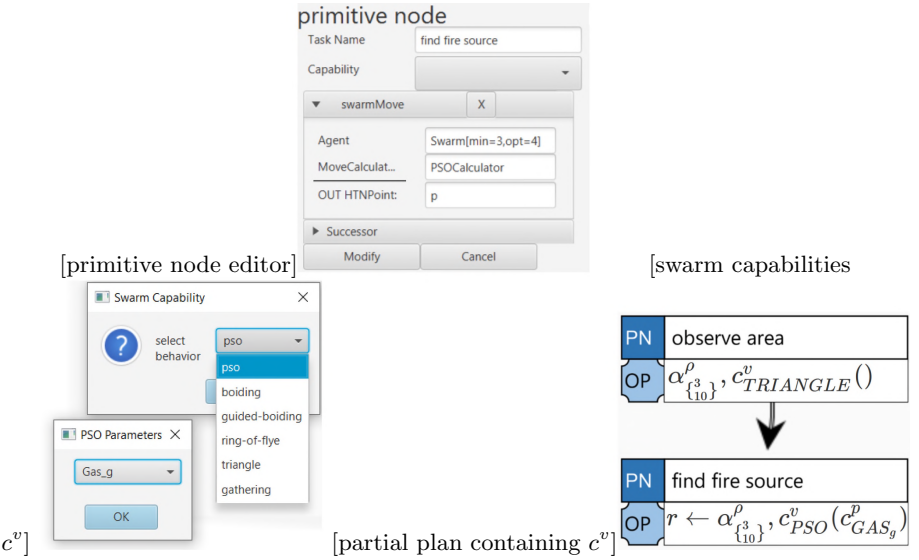


Fig. 3. The problem domain definition interface, here used for swarm capabilities.

3 Approach

In the following, we extend the possibilities available for defining the problem domain with the concepts of configurable, virtual swarm capabilities, and agent groups. Further, we describe how we extend our current graphical designer interface based on HTN, illustrate the necessary adaptations to our planning algorithm, and the transformation process for generating executable tasks from plans. Moreover, we describe how we adapt the self-awareness and self-organization approach for task allocation accordingly. We refer to the human creating a problem domain description in the form of Maple-Swarm hierarchical task networks $\mathcal{HTN}$ with our graphical tool as the *designer*. Further, we call our algorithm creating plans $\rho \in \mathcal{P}$ for the executing multipotent system as the planner $\mathbb{P}$. In our $\mathcal{HTN}$ and ρ resulting from executing $\mathbb{P}$ on the $\mathcal{HTN}$ and the world state ws (holding the current variable assignments), we use the concept of *planning* agents $\alpha^\rho \in \mathcal{A}^\rho$ to define roles of responsibility within a plan. To robots that adopt roles in ρ and that form an ensemble $\mathcal{E}$ for executing that plan at run-time, we refer to as *executing* agents $\alpha^e \in \mathcal{A}^e$ instead (Fig. 3).

3.1 Extending the Knowledge Base for Swarm Capabilities

In Maple, we already enlist possible (physical) robot capabilities $c^p \in \mathcal{C}^p$ including their necessary set of parameters and their return values as a triple $\langle c^p, \text{PAR}_{c^p}, \text{RET}_{c^p} \rangle$ in a capability knowledge base. $\mathcal{C}^p$ are such capabilities a robot can execute alone with physically available hardware, e.g., measure the concentration of a gas of type g with a GAS_g sensor. For a physical capability $c^p_{\text{MV-POS}}$,

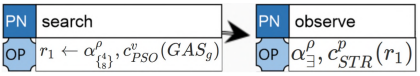


Fig. 4. Addressing $c^v_{\text{PSO}} \in C^v$ to a swarm of $\text{MIN} = 4$ and $\text{MAX} = 8$ agents and $c^p_{\text{STR}} \in C^p$ to any agent of this swarm.

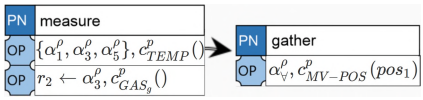


Fig. 5. Parallely addressing $c^p_{\text{TEMP}} \in C^p$ to an agent set $\{\alpha^\rho_1, \alpha^\rho_3, \alpha^\rho_5\}$ and $c^p_{\text{GAS}_g} \in C^p$ to α^ρ_3 and then $c^p_{\text{MV-POS}} \in C^p$ to all agents

that moves a robot to a certain position, e.g., the designer can find an entry defining the respective parameter $\text{PAR}_{c^p_{\text{MV-POS}}} := \langle \text{X,Y,Z} \rangle$ and the return value $\text{RET}_{c^p_{\text{MV-POS}}} := \langle \text{X,Y,Z} \rangle$ within the knowledge base. The designer can use all entries in the capability knowledge base to include them in partial plans ρ^{PART} and address them to planning agents $\alpha^\rho \in \mathcal{A}^\rho$ within the problem domain description in the $\mathcal{HTN}$. For expressing this association between capabilities and $\alpha^\rho \in \mathcal{A}^\rho$, we use *operators* (OP in our figures). We now extend this knowledge base with virtual swarm capabilities $c^v \in \mathcal{C}^v \subset C$, i.e., $c^p \in C^p \subset C$ and $C^p \cap C^v = \emptyset$ and $C^p \cup C^v = C$ by adding their respective information. This enables the designer to define ρ^{PART} addressing any capability, no matter whether it is physical or virtual (cf. Sect. 3.1, addressing a swarm-agent introduced in Sect. 3.2). Despite there is a great difference in executing a c^v instead of a c^p because all $c^v \in \mathcal{C}^v$ can only be executed by whole collectives while all $c^p \in \mathcal{C}^p$ also by particular robots alone, we enable the designer to abstract from the details when designing any ρ^{PART} for the problem domain. To include a virtual swarm capability, e.g., for executing PSO algorithm to determine the position of the highest concentration of a certain GAS_g (cf. Sect. 3.1), we thus include an entry for $c^v_{\text{PSO}} \in \mathcal{C}^v$ with the parameter $\text{PAR}_{c^v_{\text{PSO}}} := \text{GAS}_g$ and the identified position $\text{RET}_{c^v_{\text{PSO}}} := \langle \text{X,Y,Z} \rangle$ (cf. Sect. 3.1). This enables the designer to use virtual capabilities similar to physical capabilities.

3.2 Extending the Maple Domain Description Model

For creating a $\mathcal{HTN}$ and the partial plans ρ^{PART} it contains, the designer can use all elements of our extended HTN planning approach from [12], i.e., compound nodes (CN), primitive nodes (PN), world state modification nodes (WS), re-planning nodes (RP), as well as our concept for looped execution of nodes. A partial plan ρ^{PART} thus consists of nodes containing information the designer requires the system to execute, e.g., $\rho^{\text{PART}}_1 := [\text{PN}_1, \text{PN}_2, \text{WS}, \text{RP}] \in \mathcal{HTN}$. In contrast to CN (a commonly known element of HTN [6]) we use for structuring the $\mathcal{HTN}$ and for achieving situation awareness concerning the world-state during planning (cf. Sect. 3.3), all other nodes can occur in a plan ρ and thus contain instructions the multipotent system should execute. We now describe the new possibilities the designer has to define instructions in these other nodes.

Planning Agent Groups and Virtual Swarm Capabilities: In Maple, one specific plan ρ can contain one or more PN that can assign capabilities to different planning agents $\alpha^{\rho} \in \mathcal{A}^{\rho}$ in multiple *operators* (OP). Thereby, each ρ generates requirements for executing agents $\alpha^{e} \in \mathcal{A}^{e}$ to be met at run-time. We distinguish between multiple classes of planning agents. Particular planning agents $\alpha_i^{\rho} \in A_I^{\rho} \subset \mathcal{A}^{\rho}$ can be reused across the plan. To adopt the role of an α_i^{ρ} we consequently require α_i^{e} to provide all capabilities assigned to α_i^{ρ} within ρ in any node. While it was only possible to require the execution of a capability from such a specific α_i^{ρ} in Maple [12] (cf. Sect. 3.1), we now allow the designer to also specify that a *swarm of agents* $\alpha^{\rho}_{\{\text{MIN}\atop\text{MAX}\}}$, *all agents* $\alpha^{\rho}_{\forall}$, *any agent* $\alpha^{\rho}_{\exists}$, or a set of particular agents $\{\alpha_1^{\rho}, ..., \alpha_n^{\rho}\}$ need to execute a specific capability in an operator. This becomes necessary for swarm capabilities $c^{v} \in \mathcal{C}^{v}$ encapsulating collective behavior (cf. Sect. 3.1). We can not or even do not want to determine precisely how many executing agents $\alpha^{e} \in \mathcal{A}^{e}$ in an ensemble should execute a swarm capability $c^{v} \in \mathcal{C}^{v}$ at run-time. An ensemble executing $c^{v}_{\text{PSO}} \in \mathcal{C}^{v}$ (cf. Sect. 3.1), e.g., can achieve the desired emergent effect with very different swarm sizes and thus we want to decide on the number of participating entities at run-time rather than at design-time concerning the current situation the system finds itself located in. Nevertheless, there may be minimum and maximum bounds for swarm behavior to emerge at all and stay efficient [1]. For enabling the designer to define such bounds, we introduce a *swarm-agent* $\alpha^{\rho}_{\{\text{MIN}\atop\text{MAX}\}} \in A_S^{\rho} \subset \mathcal{A}^{\rho}$. This can become handy, e.g., if at least MIN and at most MAX agents should execute c^{v}_{PSO} (cf. primitive node *search* in Fig. 4). An execution agent can take the role of up to one α_i^{ρ} and additionally adopt any number of swarms-agent roles. Thus, we can also express the concept of *any-agent* $\alpha^{\rho}_{\exists} = \alpha^{\rho}_{\{1\}}$ as a specific swarm-agent. With an operator addressing a capability in ρ to the *all-agent* with $\alpha^{\rho}_{\forall} \in A^{\rho}_{\forall} \subset \mathcal{A}^{\rho}$, the designer can achieve that all i agents in an ensemble $\mathcal{E} = \{\alpha_1^{e}, ..., \alpha_i^{e}\}$ created at run-time need to execute the associated capability (A_I^{ρ}, A_S^{ρ}, and $A^{\rho}_{\forall}$ are pairwise disjunct sets). This can be useful, e.g., when all agents should gather at a dedicated position pos_1 by executing $c^{p}_{\text{MV-POS}}$ after measuring parameters of interest at different locations (cf. PN *gather* in Fig. 5). Similarly, by associating a capability with an *agent-set* $\{\alpha_1^{\rho}, ..., \alpha_n^{\rho}\} \subseteq A_I^{\rho}$, the designer can require that a concretely specified set of particular agents $\{\alpha_1^{e}, ..., \alpha_n^{e}\} \subseteq \mathcal{E}$ executes the associated capability (cf. measuring temperature with c^{p}_{TEMP} in PN *measure* in Fig. 5). Like with associating a capability to a particular planning agent α_i^{ρ} in an operator, the designer can reference to a single planning agent with the any-agent $\alpha^{\rho}_{\exists}$. Both α_i^{ρ} and $\alpha^{\rho}_{\exists}$, require that one $\alpha^{e} \in \mathcal{E}$ executes the capability at run-time. But instead of determining a particular role α_i^{ρ} at design-time that needs to execute all capabilities in ρ assigned to α_i^{ρ}, using $\alpha^{\rho}_{\exists}$ allows for any α^{e} to take the role of $\alpha^{\rho}_{\exists}$ in addition to any role it already took. This means that any one of $\{\alpha_1^{e}, \alpha_2^{e}, \alpha_3^{e}\}$ adopting the roles $\{\alpha_1^{\rho}, \alpha_2^{\rho}, \alpha_3^{\rho}\}$ later on can also execute the capabilities assigned to $\alpha^{\rho}_{\exists}$. This can also be useful when using $\alpha^{\rho}_{\{\text{MIN}\atop\text{MAX}\}}$ in plans, e.g., if after determining a point of interest with c^{v}_{PSO}, anyone of the agents that executed c^{v}_{PSO} should stream a video from that point of interest with c^{p}_{STR} (cf. PN *observe* in Fig. 4). While we introduce the swarm-, all-, set-,

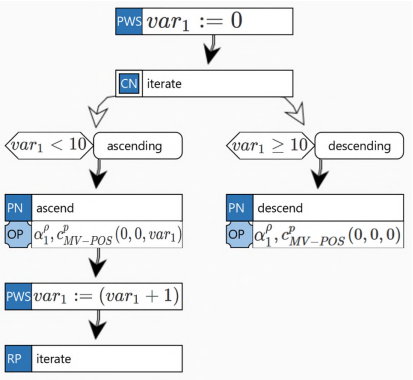


Fig. 6. Modify var_1 at planning time PWS

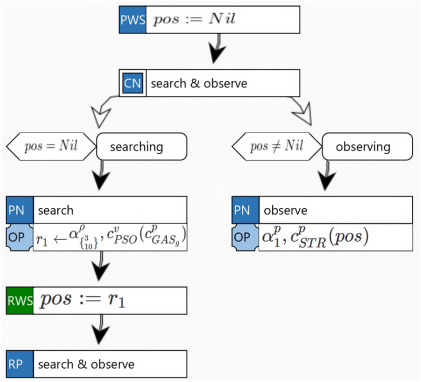


Fig. 7. Modify pos at run-time RWS

and any-agent having virtual swarm capabilities in mind, a designer can also make use of them for addressing physical capabilities $\mathcal{C}^p$. To indicate to the designer, that for executing a virtual swarm capability we require a collective and not a particular planning agent, we restrict the possibilities the designer has for addressing any $c^v \in \mathcal{C}^v$ to the respective planning agents.

Planning Variables: We further extend the concept of variables we use for expressing situations of the world state in our problem domain description, aiming for more flexibility and expressiveness. The designer now can require that values of variables update dynamically in partial plans not only during planning time but also at run-time. Moreover, we extend the way how updated variable values can be used within parameters of capabilities and in conditions, we evaluate during execution or planning. During planning time, we can update variable values only by explicitly using *planning time world-state modification nodes* (PWS) in partial plans [12]. A PWS node can contain one or multiple assignments to variables where the left side is a variable and the right side is an expression containing variables or constants, e.g., $\{var_1 := 1\}$, $\{var_2 := 2 \cdot var_1\}$, or $\{var_1 := 1, var_2 := 2\}$. This can be useful, e.g., if we want to create plans containing iterative behavior. We can achieve such by using a variable in a capability's parameter and in a condition the planner $\mathbb{P}$ evaluates during planning at the same time. If, e.g., we require an ensemble to repeat a primitive node containing $c^p_{\text{MV-POS}}$ with $par_{c^p_{\text{MV-POS}}} := \langle 0, 0, var_1 \rangle$ for 10 iterations, where var_1 is a variable we update during planning, we can achieve this by explicitly updating the value of var_1 in a PWS node (cf. Fig. 6).

We further extend our problem domain description in a way that we can also use the results of capability executions to update variable values during run-time. If we want to use variables updated that way, we need to differentiate between two cases concerning the way we want to use them in partial plans. **A)** When we use variables in primitive nodes that are updated by the executing ensemble during run-time within the same partial plan, there is no need for

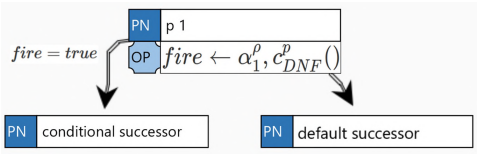


Fig. 8. IF/ELSE block evaluating the result of c^p_{DNF} that can detect fires.

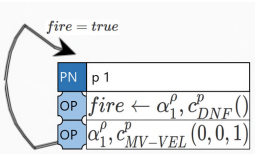
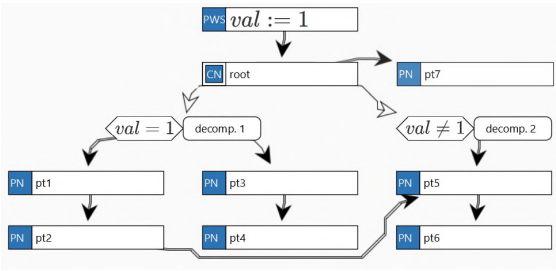


Fig. 9. WHILE block, terminated if a fire is detected.

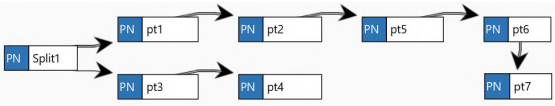
making these updates explicit within world state modification nodes. Because the ensemble executing the plan produces the new variable value itself by executing the respective capability, it is aware of that update and thus can use it in a following PN. In the partial plan in Fig. 4, e.g., we can use the result r_1 from executing c^v_{PSO} in the PN *search* as a parameter for c^p_{STR} in the PN *observe* after storing the result r_1 with a RWS node in the variable *pos* which we can use in a subsequent RP node. **B)** When we use the results of any capability's execution contained in one specific partial plan ρ_1^{PART} in another partial plan ρ_2^{PART}, we require to make the update to that variable explicit within a *run-time world state modification node* (RWS). We can use this if we do not necessarily want the ensemble executing ρ_1^{PART} to be the same than that executing ρ_2^{PART}. If, e.g., in contrast to the example in Fig. 4 we want to explicitly let another ensemble consisting of α_1^ρ execute c^p_{STR} instead of the ensemble executing c^v_{PSO}, we can store the result of c^v_{PSO} in an additional variable (POS) in a RWS node. Now, we can still access POS after finishing the value-producing plan during a subsequent re-planning that is aware of the update in PN *observe* (cf. Fig. 7). We further can use RWS to generate even more situation awareness, e.g., decide on the next PN according to the result of a capability's execution with conditional successor nodes (cf. Figs. 8 and 9). Each PN in a $\rho^{\text{PART}} \in \mathcal{HTN}$ can have any number of conditional successors assigned with variables in addition to a default successor (cf. planning variables in Sect. 3.2), evaluated by the ensemble at run-time.

3.3 Extending the Maple Planner

Executing the automated planner $\mathbb{P}$ on an $\mathcal{HTN}$ and its accompanying world state ws (that holds the current values of relevant variables), i.e., applying $\mathbb{P}$ ($\mathcal{HTN}$, ws), results in a plan ρ and a modified version of ws. Depending on the current situation represented in an up-to-date world state (updated by previous capability executions or world state modifications, cf. planning variables in Sect. 3.2), this ρ then connects partial plans (using $\oplus$) from $\mathcal{HTN}$ whose execution the designer intended to be necessary for that situation (cf. Fig. 10).



(a) $\mathcal{HTN}$ with situation aware planning



(b) plan ρ consisting of the sequences $[pt_1, pt_2, pt_5, pt_6] \oplus [pt_7]$ and $[pt_3, pt_4]$

Fig. 10. Because $val = 1$ in the world state, $\mathbb{P}$ ($\mathcal{HTN}$, ws) results in the plan $\rho = [split1] \oplus (\rho_1^{\text{PART}} \oplus \rho_4^{\text{PART}} \mid \rho_2^{\text{PART}})$, consisting of two concurrent sequences.

By evaluating conditions on variables in the world state, $\mathbb{P}$ decomposes a compound node CN into a $\mathcal{HTN}$', which is a subcomponent of the original $\mathcal{HTN}$. Each $\mathcal{HTN}$' then includes the associated partial plans, i.e., when decomposing CN *root* in Fig. 10a with the variable $val = 1$, the resulting $\mathcal{HTN}$' consists of the subordinated partial plans $\rho_1^{\text{PART}} := [pt_1, pt_2, pt_5, pt_6]$, $\rho_2^{\text{PART}} := [pt_3, pt_4]$, and a successor $\rho_4^{\text{PART}} := [pt_7]$ ($\rho_3^{\text{PART}} := [pt_5, pt_6]$ is not included). If the designer intends to have multiple concurrent plans, the designer can add multiple concurrent successors to a decomposition node. If a decomposition node with concurrent successors is encountered when running $\mathbb{P}$ ($\mathcal{HTN}$, ws), then a *split node*(e.g., *split1* in Fig. 10) connects the concurrent partial plans resulting in the plan $\rho = [split1] \oplus (\rho_1^{\text{PART}} \mid \rho_2^{\text{PART}})$, where the $\mid$ operator indicates that those partial plans can be executed concurrently. Using split nodes results in a plan consisting not only of one but multiple sequences as the output executing $\mathbb{P}$, each consisting of concatenated partial plans (cf. Fig. 10b). To make explicit how these concurrent sequences of partial plans are concatenated, a split node specifies which sequence continues the original plan (cf. double lined arrow in Fig. 10b) and which are concurrent sequences. If the CN is decomposed and has a default successor (e.g., ρ_4^{PART}), then this successor gets concatenated with the previous original plan, i.e., $\rho = [split1] \oplus (\rho_1^{\text{PART}} \oplus \rho_4^{\text{PART}} \mid \rho_2^{\text{PART}})$ in Fig. 10. The operation $\rho_1^{\text{PART}} \oplus \rho_4^{\text{PART}}$ sets the starting node of ρ_4^{PART} as the default successor of each node in ρ_1^{PART} which has no default successor yet. In contrast to CN and PWS, the other nodes PN, RWS, and RP are effectively included in a plan ρ if they occur in a partial plan ρ^{PART} that $\mathbb{P}$ selects during planning. By using a RP, the designer can enforce the generation of new plans at run-time. RP nodes hold a reference to another node of the $\mathcal{HTN}$ indicating where to start a subsequent execution of $\mathbb{P}$ at run-time with updated variables in the world state (cf. planning variables in Sect. 3.2).

3.4 Extending the Self-awareness and Market-Based Task-Allocation

We extend our local self-awareness mechanism to maintain the functionality of our market-based task-allocation mechanism introduced in [14]. While we redefine the task allocation problem in this section, we do not modify the task allocation process itself and still fall back to our constraint satisfaction and optimization-based solution from [13] and [7]. Referring to our multipotent systems reference architecture (cf. Fig. 2), each $\alpha^e \in \mathcal{A}^e$ can only provide a (physical) capability c^p and participate in a specific plan ρ requiring that capability, i.e., adopt the role of the associated α^ρ included in ρ, if all necessary hardware for c^p is connected. Thus, α^e can execute, e.g., the capability $c^p_{\text{GAS}_g}$ for measuring the concentration of GAS_g when it has a respective GAS_g connected. If this is the case, we add this capability to the set of available capabilities $C_{a^e} \subset C$ of α^e.

In contrast to physical capabilities, virtual swarm capabilities $c^v \in \mathcal{C}^v$ do not require any hardware. Instead, the *parametrization* PAR_{c^v} of a specific swarm capability referencing other capabilities $c^p \in \text{PAR}_{c^v}$ determines whether the agent can execute c^v or not and thus, if the virtual capability is available to the executing agent or not. A virtual swarm capability c^v_{PSO}, e.g., parametrized to find the source of a GAS_g (e.g., a fire) requires the physical capabilities $c^p_{\text{GAS}_g}$ and $c^p_{\text{MV-VEL}}$ (for moving with a given velocity) to be executable. Thus, a $c^v \in \mathcal{C}^v$ is only available to a α^e, if all capabilities included in the virtual capability's parameters are also available to the agent, i.e., $c^v \in C_{\alpha^e} \Leftrightarrow \forall c^p \in \text{PAR}_{c^v} \mid c^p \in C_{\alpha^e}$.

To form an ensemble $\mathcal{E}$ consisting of executing agents that are collectively able to execute a certain plan ρ, we formulate a task allocation problem. We define a task $t_{\alpha^\rho_i}$ for each different role of an identified planning agent $\alpha^\rho_i \in \mathcal{A}^\rho_I$ that is included in a plan ρ first. Thereby, we generate a set $T^\rho_I := \{t_{\alpha^\rho_i} \mid \alpha^\rho_i \in \rho\}$ of tasks we need to assign to executing agents for finally executing ρ at runtime. Besides information on how to execute ρ cooperatively within $\mathcal{E}$, which we do not further focus on here[1], we include a set C_t of required capabilities in each task's description. An executing agent α^e can adopt the role of a α^ρ_i if it has all necessary capabilities available for the respective task, i.e., we require $C_t \subseteq C_{\alpha^e}$ for the respective role's task to achieve a valid adoption. If this is the case, an executing agent α^e can participate in the market-based task allocation mechanism by generating a proposal $\text{PRO}_{\alpha^e}(t)$ for that task $t \in T^\rho_I$, cf. Eq. (1). All $\alpha^e \in \mathcal{A}^e$ then send their proposals to the plan's coordinator. This coordinator then can select one proposal for every task $t \in T^\rho_I$ generated from the planning agent roles α^ρ_i contained in the current plan ρ to achieve a valid task assignment. The coordinator can perform a valid task allocation TA for ρ if there exists an injective function f mapping each task $t_{\alpha^\rho_i}$ generated from ρ to a distinct executing agent $\alpha^e \in \mathcal{A}^e$, cf. Eq. (2). This executing agent then adopts the role of the planning agent the task was generated for.

[1] We describe how we coordinate plans containing only physical capabilities $c^p \in \mathcal{C}^p$ in [12] and how we extend that process for virtual swarm capabilities $c^v \in \mathcal{C}^v$ in [11].

$$\forall t \in T_\rho^I : \quad \text{PRO}_{\alpha^e}(t) \Leftrightarrow \mathcal{C}_t \subseteq \mathcal{C}_{\alpha^e} \tag{1}$$

$$\text{TA}(T_\rho^I) \Leftrightarrow \quad \exists_{f:T_\rho^I \to \mathcal{A}^e} \forall_{t_j \neq k \in T_\rho^I} : f(t_j) \neq f(t_k) \wedge \text{PRO}_{f(t_j)}(t_j) \wedge \text{PRO}_{f(t_k)}(t_k) \tag{2}$$

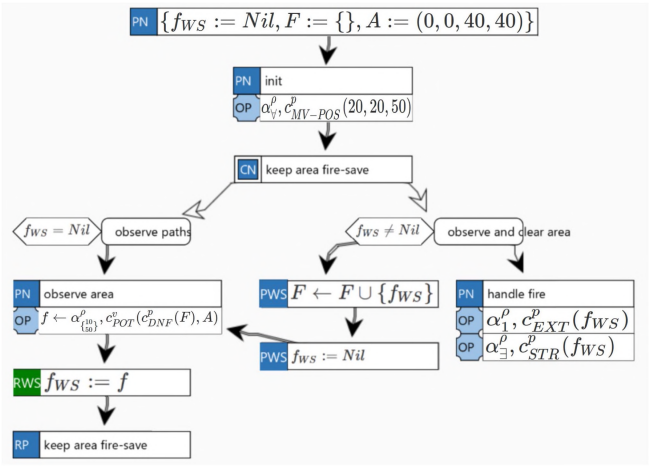
While this adaptation of the self-awareness of executing agents in the market-based task allocation mechanism can handle virtual swam capabilities, we need to perform a second adaptation to also support the agent groups introduced in Sect. 3.2. For realizing the $\alpha_\forall^\rho$, $\alpha_\exists^\rho$, and $\alpha_{\{^{\text{MIN}}_{\text{MAX}}\}}^\rho$, we need to extend the original requirements concerning the necessary capabilities for tasks $t \in T_I^\rho$ before we start the task allocation TA. If the designer addresses capabilities with $\alpha_\forall^\rho$ in the plan ρ which we can collect in the set of capabilities $C_\forall$, for creating a proposal an α^e needs to provide all these capabilities in addition to the capabilitites each task $t \in T_I^\rho$ already requires, i.e., $\text{PRO}_{\alpha^e}(t) \Leftrightarrow (\mathcal{C}_t \cup C_\forall) \subseteq \mathcal{C}_{\alpha^e}$ (cf. Eq. (1)).

Concerning adaptations of the task allocation, we fortunately can handle $\alpha_\exists^\rho$ and $\alpha_{\{^{\text{MIN}}_{\text{MAX}}\}}^\rho$ equally as we can express $\alpha_\exists^\rho$ as $c_{\{1\}}^\rho$. For every occurrence of $\alpha_\exists^\rho$ or $\alpha_{\{^{\text{MIN}}_{\text{MAX}}\}}^\rho$, we create a *swarm task* t_{SW} which we collect in a set T_{SW}^ρ. Similar to tasks $t \in T_I^\rho$ we request proposals from executing agents for all $t_{\text{SW}} \in T_{\text{SW}}^\rho$. Further, we extend the requirements for a valid task allocation to $\text{TA}(T_I^\rho) \wedge \text{TA}(T_{\text{SW}}^\rho)$, where $\text{TA}(T_{\text{SW}}^\rho)$ is valid if we have at last MIN proposals from distinct executing agents for every task $t_{\text{SW}} \in T_{\text{SW}}^\rho$ (cf. Eq. (2)). To select a range of MIN and MAX agents for every $t_{\text{SW}} \in T_{\text{SW}}^\rho$, we can optionally accept any further proposal for the respective task from a distinct agent until we reach the respective limit of MAX.

4 Proof of Concepts

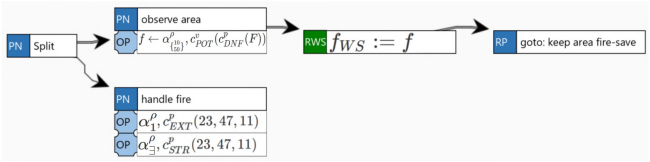
We demonstrate the new possibilities of Maple-Swarm within an exemplary $\mathcal{HTN}$ consisting of partial plans $\rho_1^{\text{PART}}, \ldots, \rho_4^{\text{PART}}$ and the plans resulting in different situations for our motivating example in Fig. 11.

In a first partial plan $\rho_1^{\text{PART}} := [\text{PWS}_1, \text{PN}_1]$ we include in $\mathcal{HTN}$ during the designing process, we initialize the relevant variables in the world state (PWS_1 sets variables f_{WS} to Nil, initializes the area of interest A where $\langle 0, 0, 40, 40 \rangle$ defines x and y coordinates as well as length and width, and F to $\{\}$) and direct the whole ensemble to the center of the forest at $\langle 20, 20 \rangle$ which we want to survey in an altitude of 50 m. We achieve this by using the physical capability $c_{\text{MV-POS}}^p$ with the parameter $\langle 20, 20, 50 \rangle$ and addressing the agent group $\alpha_\forall^\rho$ (commanded in the respective operator included in PN_1). If we do not know any fires located in the forest (i.e., $p_{fire} = Nil$), we design another partial plan $\rho_2^{\text{PART}} := [\text{PN}_2, \text{RWS}_2, \text{RP}_2]$ to let a swarm of agents $\alpha_{\{^{10}_{50}\}}^\rho$ consisting of a minimum of 10 and a maximum of 50 agents execute a virtual swarm capability to equally distribute in the area of interest (A) with the potential field algorithm



(a) $\mathcal{HTN}$ for the motivating example

(b) resulting plan if initially no fire location is known (i.e., $f_{ws} = Nil$), concatenating $\rho_1^{\text{PART}} \oplus \rho_2^{\text{PART}}$



(c) resulting plan if a fire location is known (i.e., $f_{ws} = Nil$), consisting of the concurrent partial plans ρ_3^{PART} and ρ_4^{PART}

Fig. 11. An example $\mathcal{HTN}$ consisting of situation-aware partial plans for handling the fire-fighter scenario from Sect. 1 including possible plans resulting from executing $\mathbb{P}$ ($\mathcal{HTN}$).

encapsulated in c^{v}_{POT} in PN_2. We assume, that the swarm can autonomously adapt the altitude for gaining surveillance quality according to the amount of swarm members like it is proposed to be achievable in [20]. We can achieve such behavior with an appropriate implementation of the respective swarm capability c^{v}_{POT} (we explain how we can achieve this in the accompanying publication concerning the execution of swarm capabilities [11]). In that partial plan, we use the capability c^{p}_{DNF} for detecting new fires (i.e., such not already included in F) on the ground as the parameter of c^{v}_{POT} to return the position of a fire $f := \langle f_x, f_y, 0 \rangle$ as soon as one member of the swarm detects a fire. Detecting a fire then causes an update of the world state in RWS_2 that sets the variable f_{ws} to the result of c^{v}_{POT}, i.e., $f_{ws} := f$, followed RP_2 referencing the only CN in the $\mathcal{HTN}$ (keep ara fire-save). If the ensemble is aware of a fire, i.e., the world state holds a respective entry and $f_{ws} \neq \text{NIL}$ and contains the location of the fire (cf. Fig. 11a), we design two concurrent partial plans and $\rho_3^{\text{PART}} := [\text{PN}_3]$ and $\rho_4^{\text{PART}} := [\text{PWS}_{4\text{-A}}, \text{PWS}_{4\text{-B}}, \text{PWS}_1, \text{PN}_1]$ we want the ensemble to execute in that situation. In ρ_3^{PART}, we address α^{ρ}_1 to execute a physical capability c^{p}_{EXT} for extinguishing the fire at the identified location and $\alpha^{\rho}_{\exists}$ to should stream a video from that execution to the user by executing a respective capability c^{p}_{STR}, $\text{PAR}_{c^{p}_{\text{STR}}} := f_{ws}$ which we include in an respective operator in PN_3. We can thus let the system decide with respect

to the current availability of capabilities in the ensemble whether one executing agent α^e is sufficient for executing that plan (i.e., $\alpha_1^\rho = \alpha^\exists$) or two agents are used instead. As parameter for both, c^p_{STR} and c^p_{EXT}, the planning process generates a copy of the concrete position of the fire, e.g., if $f_{ws} := \langle 23, 47, 11 \rangle$, we use the parameter $\text{PAR}_{c^p_{\text{EXT}}} := \langle 23, 47, 11 \rangle$ and $\text{PAR}_{c^p_{\text{STR}}} := \langle 23, 47, 11 \rangle$. We need that copy because in an the concurrent partial plan ρ_4^{PART}, we add the identified location of the fire f_{ws} to a set of known fires F in $\text{PWS}_{4\text{-A}}$ and then reset f_{ws} to Nil in $\text{PWS}_{4\text{-B}}$ before an other ensemble again executes *observe area* in the concatenated partial plan ρ_2^{PART}. Given this problem domain description, executing $\mathbb{P}$ on $\mathcal{HTN}$ and ws thus results in a plan consisting of $\rho_1^{\text{PART}} \oplus \rho_2^{\text{PART}}$ if $f_{\text{WS}} = Nil$ (cf. Fig. 11b) and in a plan consisting of ρ_3^{PART} concurrent to ρ_4^{PART} if $f_{\text{WS}} \neq Nil$ (cf. Fig. 11c). Besides this exemplary $\mathcal{HTN}$, we demonstrate the functionality of our approach with video materials and provide our application including source files for the presented examples from previous sections on GitHub[2].

5 Related Work

Some research already exists focusing on the problem of task orchestration for collectives, ensembles, aggregates, or swarms. A framework providing a scripting language for multi-vehicle networks is Dolphin [17]. With Dolphin, a human operator is able to define tasks for particular robots and teams of robots without explicit knowledge of the concrete implementation of these tasks' execution. While a user can define tasks for pre-formed robot teams with Dolphin, it does not support a possibility for exploiting emergent effects of collective behavior like, e.g., swarm algorithms can deliver. Further, Dolphin does not include the possibility for online and situation-aware re-planning that can generate new tasks at run-time as we support in Maple-Swarm. PaROS [3] is another multi-robot tasks orchestration framework. It introduces primitives for collectives the user can define tasks with and let them distribute within a swarm of UAV. Unfortunately, only homogeneously equipped UAV are in the focus of PaROS and there is no support for multi-robot systems in general. While PaROS does support some promising abstractions for encapsulating certain swarm behavior in tasks for groups of UAVs, it does not aim at interconnecting those tasks in complex programs with parallel, concurrent, alternating, or iterated execution of different swarm algorithms we aim for in Maple-Swarm. Further, there is no feature providing situation-awareness and run-time task generation. With TeCola [16], users can program missions for heterogeneous teams of robots on an abstract level. By abstracting the robots and capabilities of robots as services available to the user, TeCola reduces complexity for coordinating ensembles. TeCola eases the programming of robot teams with primitives for abstracting robots in teams and missions but still requires fine-grained management of those during task specification. Neither collective behavior achieved by swarm algorithms nor situation-aware task generation is supported by TeCola. Voltron [19] provides a task orchestration framework for robot teams. While the authors can achieve

[2] Materials on https://github.com/isse-augsburg/isola2020-maple-swarm.git.

the abstraction of particular robot actions including parallel task execution, scaling, and concurrent execution, by introducing so-called team-level programming, they lose the ability for controlling and specifying tasks for particular robots. Up to now, with Voltron a user can not specify collective behavior in the form of swarm algorithms. While Voltron does include a mechanism to compensate for failures at run-time, e.g., to maintain the execution of once-defined tasks, it does not support other situation-aware modifications of missions. There is no possibility for an autonomous generation of tasks at run-time like we provide with re-planning in Maple-Swarm. Recapitulating the findings in the literature, we can see that up to now there exists no task orchestration framework supporting all features we integrate into Maple-Swarm. While all presented approaches deliver benefits for programming collectives, each lacks some aspects that are of great relevance in our opinion.

6 Conclusion

Performing task orchestration for multi-robot systems is complicated, especially for domain-only experts. In this paper, we propose our approach for easing this by extending the task definition layer of our multi-agent script programming language for multipotent ensembles with virtual swarm capabilities encapsulating collective behavior. We, therefore, extended our current approach Maple concerning the graphical task designer interface, the automated planner, and the market-based task allocation mechanism including the local self-awareness functionality for every robot to Maple-Swarm. Users are now able to address tasks not only to particular robots but whole ensembles. Thereby, users can make use of collective adaptive behavior, e.g., of swarm behavior and useful emergent effects arising thereof. We demonstrated the new possibilities in examples as well as in the proof of concepts for a fire-fighter case study we provide online. Our next steps include the integration with our reference implementation for a multipotent system that can execute swarm capabilities with mobile robots.

Acknowledgement. The authors would like to thank all reviewers for their valuable suggestions.

References

1. Barca, J., Sekercioglu, Y.: Swarm robotics reviewed. Robotica **31**, 345–359 (2013)
2. Daniel, K., Dusza, B., Lewandowski, A., Wietfelds, C.: Airshield: a system-of-systems MUAV remote sensing architecture for disaster response. In: Proceedings of 3rd Annual IEEE Systems Conference (SysCon) (2009)
3. Dedousis, D., Kalogeraki, V.: A framework for programming a swarm of UAVs. In: Proceedings of the 11th Pervasive Technologies Related to Assistive Environments Conference, pp. 5–12 (2018)
4. Duarte, M., Costa, V., Gomes, J., et al.: Evolution of collective behaviors for a real swarm of aquatic surface robots. PLoS ONE **11**(3), 1–25 (2016)

5. Erol, K., Hendler, J., Nau, D.S.: HTN planning: complexity and expressivity. AAAI **94**, 1123–1128 (1994)
6. Georgievski, I., Aiello, M.: An overview of hierarchical task network planning (2014). CoRR abs/1403.7426, http://arxiv.org/abs/1403.7426
7. 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)
8. Koenig, S.: Agent-centered search. AI Mag. **22**(4), 109 (2001)
9. Kosak, O.: Facilitating planning by using self-organization. In: IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 371–374 (2017)
10. Kosak, O.: Multipotent systems: a new paradigm for multi-robot applications. Org. Comp. Doc. Dis. Coll. **10**, 53 (2018). Kassel university press GmbH
11. Kosak, O., Bohn, F., Eing, L., et al.: Swarm and collective capabilities for multipotent robot ensembles. In: 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (2020)
12. 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)
13. Kosak, O., Wanninger, C., Angerer, A., et al.: Decentralized coordination of heterogeneous ensembles using jadex. In: IEEE 1st International Workshops on Foundations and Appl. of Self* Systems (FAS*W), pp. 271–272 (2016). https://doi.org/10.1109/FAS-W.2016.65
14. 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)
15. Kosak, O., Wanninger, C., Hoffmann, A., Ponsar, H., Reif, W.: Multipotentsystems: combining planning, self-organization, and reconfiguration inmodular robot ensembles. Sensors **19**(1), 17 (2018)
16. Koutsoubelias, M., Lalis, S.: Tecola: a programming framework for dynamic and heterogeneous robotic teams. In: Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, pp. 115–124 (2016)
17. Lima, K., Marques, E.R., Pinto, J., Sousa, J.B.: Dolphin: a task orchestration language for autonomous vehicle networks. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 603–610. IEEE (2018)
18. Lorenz, R.D., Turtle, E.P., Barnes, J.W., et al.: Dragonfly: a rotorcraft lander concept for scientific exploration at titan. Johns Hopkins APL Tec. Dig. **34**, 374–387 (2018)
19. Mottola, L., Moretta, M., Whitehouse, K., Ghezzi, C.: Team-level programming of drone sensor networks. In: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, pp. 177–190 (2014)
20. Villa, T.F., Gonzalez, F., Miljievic, B., Ristovski, Z.D., Morawska, L.: An overview of small unmanned aerial vehicles for air quality measurements: present applications and future prospectives. Sensors (Basel, Switzerland) **16**(7), 1072 (2016)
21. 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

22. Wolf, B., Chwala, C., Fersch, B., et al.: The scalex campaign: scale-crossing land surface and boundary layer processes in the tereno-prealpine observatory. Bull. Am. Meteorol. Soc. **98**(6), 1217–1234 (2017)
23. Zhang, Y., Wang, S., Ji, G.: A comprehensive survey on particle swarmoptimization algorithm and its applications. Math. Prob. Eng. (2015)