A Symbiotic Orchestration Module for Multi-agent Collaboration in Disaster Response Scenarios

Thomas Theodoridis
Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH), Thessaloniki, Greece
tomastheod@iti.gr

George Katsikas
Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH), Thessaloniki, Greece
g.katsikas@iti.gr

Nicholas Vretos*
Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH), Thessaloniki, Greece
vretos@iti.gr

Petros Daras
Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH), Thessaloniki, Greece
daras@iti.gr

ABSTRACT

This paper presents the Symbiotic Orchestration Module, which facilitates the collaboration of smart agents in disaster response scenarios. By effectively orchestrating the actions of different agents in critical situations towards a common goal, it enhances the individual capabilities of the agents and unlocks new possibilities that are not available when agents act isolated. To achieve this, the Symbiotic Orchestration Module is composed of four sub-modules: a) the Mission Controller, which is responsible for keeping track of ongoing missions, agent allocations and for handling non-collaborative missions, b) the Symbiotic Operation Control Module, which handles collaborative missions proposed by the system, c) the Task Allocation Module, which automatically assigns available robots to incoming missions based on robot capabilities and mission requirements, and d) the Task Recognition and Optimal Sequencing Module, which is responsible for recognizing opportunities for agent collaboration and for system-wide goal optimization.

Keywords

Symbiotic controller, multi-robot collaboration, first responders, disaster response

INTRODUCTION

First responders are often involved in life-threatening situations when called upon to provide their services in natural or man-made disasters. In these situations, time is critical for saving human lives and the ability to quickly get an overview of the situation and take immediate actions in extremely dynamic environments is very important. Autonomous cybernetic assistants (i.e. smart agents, such as UAVs and UGVs) can provide significant benefits to first responder teams in such situations, as they allow them to quickly get an overview of the area of interest, or explore dangerous areas, such as inside flaming buildings ready to collapse, searching for survivors. However, by operating isolated in single-task missions, the smart agents can never go beyond their individual capabilities.

A module that could orchestrate the actions of smart agents with different capabilities towards a common goal, would be able to offer disaster response teams enhanced capabilities that surpass the individual capabilities of each agent. For example, an indoor UAV and a UGV equipped with a robotic arm can collaboratively explore a partially collapsed building, where some of the doors are stuck half-open, but not quite enough for the UAV to enter

*corresponding author

these rooms. The UGV can open these doors just enough with its arm so the UAV can enter the rooms and quickly explore them. By taking advantage of such collaborative efforts by the smart agents, first responder teams could quickly obtain an assessment of the situation.

Several symbiotic approaches involving smart agents have been presented in the literature, including in areas such as automated multi-agent search in unknown environments (Guruprasad and Ghose 2010), energy foraging (Kernbach et al. 2008), decentralized path planning and motion coordination of automated guided vehicles (Draganjac et al. 2016) and cooperative manipulation and transportation of different payload (Ebel et al. 2020), (Lee et al. 2016), (Hichri, Adouane, et al. 2016), (Hichri, Fauroux, et al. 2019). This paper presents the Symbiotic Orchestration Module, which facilitates the collaboration of smart agents in disaster response scenarios. To achieve this, the module consists of four distinct sub-modules: a) the Mission Controller, which supervises all running missions and executes single-agent missions, b) the Symbiotic Operation Control Module, which executes symbiotic missions, c) the Task Allocation Module, which performs the best matching between available robots and incoming missions, and d) the Task Recognition and Optimal Sequencing Module, which recognizes agent collaboration opportunities and performs system-wide goal optimization. Each of the aforementioned modules is presented in the following sections, while the conclusions are presented in the last section.

THE MISSION CONTROLLER

The main functionalities of the Mission Controller (MC) are a) to keep track of ongoing missions and smart agent allocations, b) to report the status of all ongoing missions and c) to handle the execution of single-agent missions proposed by the system. The general architecture of this component, including its interactions with other components, can be seen in Figure 1.

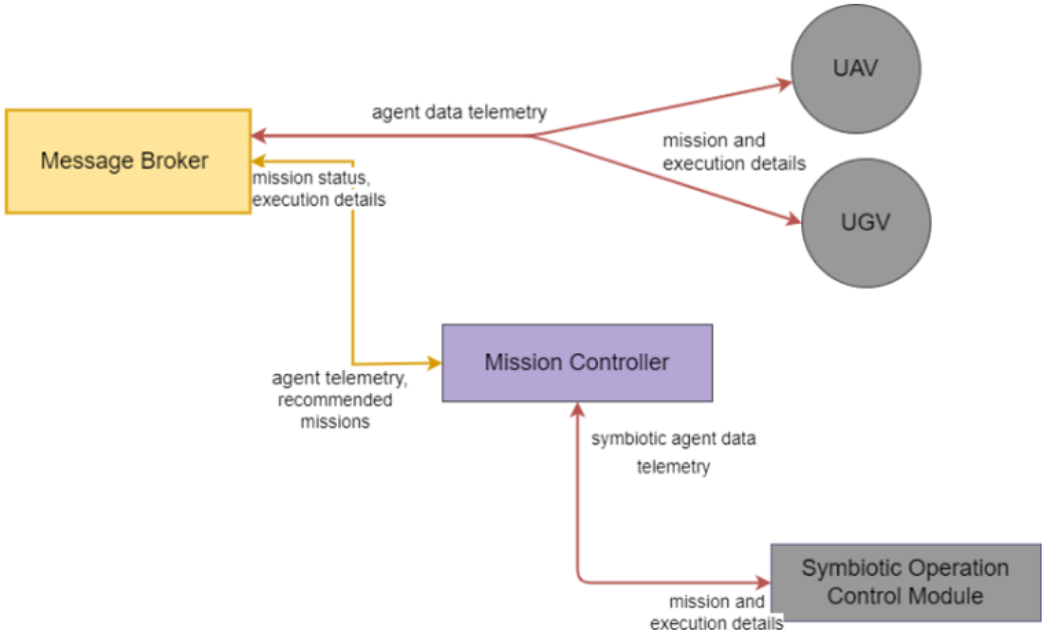

Figure 1. The Mission Controller architecture.

Upon system initialization, the Mission Controller (purple component in Figure 1) receives agent registration messages through the Kafka Message Broker (appearing in yellow in Figure 1), so that it knows all the agents available to the system, as well as their capabilities, in order to keep track of them. After registration, the agents (gray circles) start sending status messages to MC in regular intervals, again through the Message Broker, so that MC knows their state and readiness to execute any incoming missions. Upon receiving a mission request through the Message Broker, MC first determines whether it corresponds to a single-agent mission or a multi-agent symbiotic mission based on the number of agents involved. In the former case, it initiates a new single-agent mission by communicating with the agent involved in it. The MC breaks down the mission execution details into the appropriate set of commands the smart agent understands. The mission details also include which sensors and data streams are needed for this particular mission. The smart agent, after receiving this information, begins executing the commands in the given sequence and provides regular updates to the system regarding its state, its

position, battery level, etc. The MC takes into account these agent status messages in order to produce its own mission status messages, which are consumed by other modules in the architecture through the Message Broker. Mission status messages include information such as the mission status (e.g., ongoing), the completion percentage, if the agent is running low on battery or if the communication with the agent has been lost. In the multi-agent case, MC initiates a new symbiotic mission, passes its execution details to the Symbiotic Operation Control Module (SOCM) for execution, and monitors its progress through status messages produced by SOCM. In this scenario, SOCM is responsible for communicating with the smart agents and issuing the appropriate commands.

Since the proposed module is designed to operate in challenging environments, there is always the possibility of temporarily or permanently losing the communication with the smart agents. Short-term communication loss can be adequately handled by the proposed messaging component (Message Broker), as the messages intended for the agents are not lost, but remain stored and ready to be delivered as soon as the communication resumes. In the case of long-term communication loss, the appropriate actions to be taken are defined centrally by higher level modules within the INTREPID system. Since the system is composed of many other modules as well (e.g., Path Planning Module), the directive must be given centrally and not by each module in isolation. Therefore, based on the active policy in the current deployment of the system, the smart agents are expected to follow this policy when they detect a long-term communication loss. Depending on the scenario, the policy may dictate for instance that the agents go to the last known communication point or to the starting position if possible.

In order to be able to respond rapidly to proposed missions from the system, the Mission Controller keeps track of important agent parameters even if the agents are not assigned to any missions. By regularly receiving telemetry messages from all smart agents logged into the platform, MC is able to know not only their availability for any incoming missions, but also their ability to successfully accomplish the mission (e.g., their battery level is sufficient to complete the mission). Table 1 presents an example of what a single-agent mission looks like. The mission has a unique identifier, the type of task that should be performed, a priority score indicating which missions should be executed first in case of conflicts, the agents involved, any arguments that should be given to the agents in order to complete the mission and the type of data the agents need to capture and transmit while performing the mission.

Table 1. An example of a single-agent mission.

Mission ID	8
Task	Exploration
Priority	0.7
Agents	Any UAV

Arguments	{
	Coordinate system: 'local'
	Path: $[(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)]$
	}

Sensors and Streams	RGB-D Sensor: Colour and Depth images

THE SYMBIOTIC OPERATION CONTROL MODULE

The Symbiotic Operation Control Module (SOCM) is responsible for handling any symbiotic missions proposed by the system. As discussed in the previous section, the Mission Controller passes the execution details of symbiotic missions to SOCM and just supervises their progress. Upon receiving the execution details of the mission, SOCM is responsible for communicating with each agent involved in the given mission and issuing the appropriate commands to each one of them, in order to successfully complete the mission.

When the platform proposes a collaborative mission, the MC initiates a new symbiotic mission and instructs SOCM to take control of it. The MC then continues to supervise the mission and its progress, producing periodic mission status updates. In this instance, however, it is SOCM that gathers the status messages produced by the agents involved in the mission into one final status message for the mission, which is passed on to MC in order to update its state regarding the mission and send it to the Message Broker for all interested modules to consume. For example, if

there are 2 agents in a symbiotic mission and have executed 40% and 60% of their respective commands, then SOCM would report a mission completion progress of 50% to MC. The overall mission status reported by SOCM would also include agent-related information, such as if any agent is low on battery or has lost communication with the system. The communication between SOCM and the agents, although not shown in Figure 2, is the same as the previous case; the messages are relayed through the Message Broker.

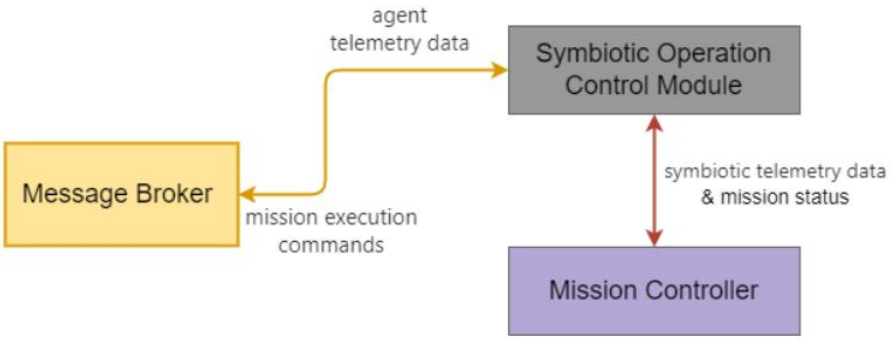

Figure 2. The Symbiotic Operation Control Module architecture.

From this point onward, SOCM is responsible for translating the mission execution details into the appropriate command sequences for each agent, handling any peculiarities of symbiotic missions as well. For instance, when a UGV needs to open a door (with its robotic arm) so that a UAV can pass through and quickly explore the area, SOCM has to make sure the door is fully open before instructing the UAV to go through. Each command sequence consists of a list of predefined instructions that the smart agents can understand and execute. The mission execution instruction set is presented in Table 2. The first four commands (*Goto, Follow, Scan Area and Move*) involve the agents moving from one position to another, either using a given path or without any, by using their navigation capabilities in order to avoid collisions. The next command, *Turn*, involves a stationary 360 degree turn by the smart agents and was included to tackle cases where the first responders need to quickly assess the surrounding environment. By taking into account the sensor streaming capabilities of the agents, this command can be combined for instance with RGB, depth or point cloud data streaming to the platform, so that other modules can quickly produce paths for the agents and assess the environment. The next four command categories (*Takeoff / Land, Open Door, Press Button / Switch and Pickup / Putdown*) involve the agents interacting with their surroundings in various ways. The *Takeoff / Land* category is UAV specific, while the *Open Door* and *Pickup / Putdown* categories are UGV specific. All command messages have a common interface that they adhere to, despite having different number and types of arguments, so that they can be effectively parsed by any agent. Finally, it is worth pointing out that this version of the mission execution instruction set is capable of facilitating both single-agent and collaborative missions.

Table 3 presents an example of a symbiotic mission, which employs a UAV and a UGV agent and has two objectives. First, both agents must explore the area using the paths given by the system. After this part is finished, the UAV must move to the position of the UGV and land on top of it, at the recharging bay, in order to keep its battery level high. Upon receiving this mission, the MC passes its execution details to the SOCM and starts supervising the mission progress. The SOCM analyzes the mission details and starts issuing the appropriate commands to the agents. First, SOCM communicates the needed sensors and streams to the agents. Then a *Goto* command is given to both agents with the given path specifications. After determining that both agents have completed this part of the mission, based on their status updates and positions, SOCM issues a *Move* command to the UAV towards the direction of the UGV. Finally, SOCM issues a *Land* command to the UAV, targeting the position of the UGV, with the special option to land on top of another agent set to true. After successfully completing the mission, the MC updates the mission status to *Completed* and also its agent allocation table.

TASK ALLOCATION

The Task Allocation component allows the system to be able to automatically assign available agents to incoming missions, when no specific agents are identified in the mission details. To accomplish this, each incoming mission to the system must have a list of specifications that can be matched to agent sensorial or other capabilities (e.g., a mission may require streaming of chemical sensor data that are only available from a single agent). On the other

Table 2. The mission execution instruction set.

Command	Description
Goto	A UxV must reach a destination following a given path.
Follow	A UxV must follow another UxV.
Scan Area	A UxV must scan the surrounding area.
Move	A UxV must move a small distance away from its current position without a given path.
Turn	A UxV must perform a 360 degree turn.
Takeoff / Land	A UAV must perform a takeoff / landing.
Open Door	A UGV must open a door with its arm.
Press Button / Switch	A UxV must press a button / switch.
Pickup / Putdown	A UGV must pick up / put down an item using its arm.

Table 3. An example of a symbiotic mission.

Mission ID	44
Task	Exploration, Recharging
Priority	0.9
Agents	UAV_1, UGV_1

Arguments	UGV_1: {
	Coordinate system: 'local'
	Path: $[(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)]$
	}, UAV_1: {
	Coordinate system: 'local'
	Path: $[(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)]$
	Recharge on UGV_1
	}

Sensors and Streams	UGV_1: { RGB-D Sensor: Colour and Depth images }, UAV_1: { RGB-D Sensor: Point cloud }

hand, each agent that is logged into the platform must also make its capabilities known to the system. When both of these conditions are met, the Task Allocation component can provide fast, near-optimal allocation solutions.

The first version of the component is based on the game-theoretic approach of (Li et al. 2020), which tackles the multi-robot task allocation problem. The authors showed in their work that the algorithm scales up to thousands of tasks and robots, achieves the same utility scores as well-known alternative methods, such as the Distributed Stochastic Algorithm (Fitzpatrick and Meertens 2003; Zhang, Wang, Xing, et al. 2005; Zhang, Wang, and Wittenburg 2002) and the Branch-and-Bound Fast-Max-Sum algorithm (Macarthur et al. 2011) when the number of tasks is low, while it achieves better utility scores when the number of tasks increases. At the same time, as shown in (Li et al. 2020), it consistently surpasses the aforementioned alternative approaches in execution time.

The algorithm works iteratively, in rounds, until no better solution can be found or some other termination criterion has been activated, such as a time constraint. More specifically, let R represent the requirements for m given tasks:

$$R = \begin{bmatrix} \rho_1^1 & \cdots & \rho_K^1 \\ \vdots & \ddots & \vdots \\ \rho_1^m & \cdots & \rho_K^m \end{bmatrix}, \rho_k^j \in \{0, 1\}$$

and C represent the capabilities of n given robots:

$$C = \begin{bmatrix} c_1^1 & \cdots & c_K^1 \\ \vdots & \ddots & \vdots \\ c_1^n & \cdots & c_K^n \end{bmatrix}, c_k^i \in \mathbb{Z}^+$$

Let G_j denote the group of robots allocated to task t_j and G_0 denote the group of robots that are currently unallocated. In the beginning, each robot r_i starts as unallocated (it belongs to G_0). Finally, let U_j denote the task utility. The task utility function can be an arbitrary function that takes into account the task requirements and the robot capabilities and produces a fitness score, indicating how well the robots allocated to this task satisfy its requirements. A simple and intuitive choice for the utility function is the following:

$$U_j = \left[\max_i c_1^i, \cdots, \max_i c_K^i\right] \cdot \left[\rho_1^j, \cdots, \rho_K^j\right]^T$$

where i traverses the indices of the robots belonging to G_j. In other words, this utility function is the dot product of the maximum capability offered by the robots assigned to perform task t_j and the requirements of the task. In each round, the algorithm first computes the marginal contribution of robot r_i to task t_j, i.e. the difference between the task utility when the robot is part of the task and when it is not. The next step is to compute the movement value of each robot from its current task to all other tasks. The movement value represents the reward gain for robot r_i when switching from its current task to another, and is defined as the difference between the marginal contribution to its current task j and the marginal contribution to any other task h. Next, if the maximum movement value across all tasks for robot r_i is positive, the robot can improve the overall utility of the system by switching tasks and joining G_h, where h is the task that the maximum was achieved for. In this way, all robots that can improve the overall utility of the system, are candidates for switching tasks. In the next step, the system gathers all movement values for each task and determines which robots produced the biggest values. These robots receive an acceptance signal by the system, while all other movement proposals are rejected. Finally, each robot that received an acceptance from task t_h moves to the group G_h. At this point, one iteration of the algorithm has been completed. This process continues until no positive movement values exists, or some other stopping criterion is reached, such as a specific utility threshold or a given time budget has been exceeded. Finally, it is worth pointing out that the algorithm cannot be halted due to a deadlock, as any resulting ties are resolved by randomly selecting one of the candidates.

Listing 1 presents an example evaluation of the task allocation algorithm. Matrix R indicates that there are 2 tasks that have 5 requirements, while matrix C indicates that there are 4 available robots with the corresponding capability scores to these requirements. Furthermore, the algorithm is parametrized to allocate up to 2 robots per task and run for a maximum of 50 rounds. The first batch of messages indicate the marginal contributions of the robots at the first round. The (-) sign indicates the task utility when the robot is not part of the task and the (+) indicates the utility when the robot is. The next batch of messages indicate the movement values of the robots. If a robot has a positive movement value, it is proposed by the system that it moves to the task which achieves its maximum movement value. In the next step, the system gathers all proposed moves and selects for each task the ones that maximize its utility. Finally, the candidate robots that received acceptance messages are assigned to new tasks. The process continues for another 2 rounds and terminates, as there are no new proposals for movement. The final allocation is that robots 0 and 1 are assigned to task 1 and robots 2 and 3 are assigned to task 0, achieving a total system utility of 39.

```
# task requirements
R = [[1, 1, 1, 0, 0],
     [1, 0, 0, 1, 1]]

# robot capabilities
C = [[0, 0, 0, 7, 6],
```

```
     [4, 3, 8, 8, 4],
     [0, 9, 8, 0, 0],
     [4, 9, 0, 7, 0]]

# <-- start of round 1 -->
Robot 0, Task 0 --> task score (-): 0.0, task score (+): 0.0,  contribution: 0.0
Robot 0, Task 1 --> task score (-): 0.0, task score (+): 13.0, contribution: 13.0
Robot 1, Task 0 --> task score (-): 0.0, task score (+): 15.0, contribution: 15.0
Robot 1, Task 1 --> task score (-): 0.0, task score (+): 16.0, contribution: 16.0
Robot 2, Task 0 --> task score (-): 0.0, task score (+): 17.0, contribution: 17.0
Robot 2, Task 1 --> task score (-): 0.0, task score (+): 0.0,  contribution: 0.0
Robot 3, Task 0 --> task score (-): 0.0, task score (+): 13.0, contribution: 13.0
Robot 3, Task 1 --> task score (-): 0.0, task score (+): 11.0, contribution: 11.0

Robot 0, Task -1 --> 0 gain 0.0
Robot 0, Task -1 --> 1 gain 13.0
Robot 0 proposes move -1 --> 1 (gain: 13.0)

Robot 1, Task -1 --> 0 gain 15.0
Robot 1, Task -1 --> 1 gain 16.0
Robot 1 proposes move -1 --> 1 (gain: 16.0)

Robot 2, Task -1 --> 0 gain 17.0
Robot 2, Task -1 --> 1 gain 0.0
Robot 2 proposes move -1 --> 0 (gain: 17.0)

Robot 3, Task -1 --> 0 gain 13.0
Robot 3, Task -1 --> 1 gain 11.0
Robot 3 proposes move -1 --> 0 (gain: 13.0)

Task 0, robot candidate: 2
Task 1, robot candidate: 1

Robot 2 has been assigned to task 0
Robot 1 has been assigned to task 1
# <-- end of round 1 -->

--------------------------------------------------
Round 1, time: 0.006 seconds, total utility: 33.0
--------------------------------------------------

# <-- start of round 2 -->
Robot 0, Task 0 --> task score (-): 17.0, task score (+): 17.0, contribution: 0.0
Robot 0, Task 1 --> task score (-): 16.0, task score (+): 18.0, contribution: 2.0
Robot 1, Task 0 --> task score (-): 17.0, task score (+): 21.0, contribution: 4.0
Robot 1, Task 1 --> task score (-): 0.0,  task score (+): 16.0, contribution: 16.0
Robot 2, Task 0 --> task score (-): 0.0,  task score (+): 17.0, contribution: 17.0
Robot 2, Task 1 --> task score (-): 16.0, task score (+): 16.0, contribution: 0.0
Robot 3, Task 0 --> task score (-): 17.0, task score (+): 21.0, contribution: 4.0
Robot 3, Task 1 --> task score (-): 16.0, task score (+): 16.0, contribution: 0.0

Robot 0, Task -1 --> 0 gain 0.0
Robot 0, Task -1 --> 1 gain 2.0
Robot 0 proposes move -1 --> 1 (gain: 2.0)

Robot 1, Task 1 --> 0 gain -12.0
Robot 1, Task 1 --> 1 gain 0.0
Robot 1 proposes stay @ 1 (gain: 0.0)

Robot 2, Task 0 --> 0 gain 0.0
Robot 2, Task 0 --> 1 gain -17.0
Robot 2 proposes stay @ 0 (gain: 0.0)
```

```
Robot 3, Task -1 --> 0 gain 4.0
Robot 3, Task -1 --> 1 gain 0.0
Robot 3 proposes move -1 --> 0 (gain: 4.0)

Task 0, robot candidate: 3
Task 1, robot candidate: 0

Robot 3 has been assigned to task 0
Robot 0 has been assigned to task 1
# <-- end of round 2 -->

------------------------------------------------
Round 2, time: 0.014 seconds, total utility: 39.0
------------------------------------------------

# <-- start of round 3 -->
Robot 0, Task 0 --> task score (-): 21.0, task score (+): 21.0, contribution: 0.0
Robot 0, Task 1 --> task score (-): 16.0, task score (+): 18.0, contribution: 2.0
Robot 1, Task 0 --> task score (-): 21.0, task score (+): 21.0, contribution: 0.0
Robot 1, Task 1 --> task score (-): 13.0, task score (+): 18.0, contribution: 5.0
Robot 2, Task 0 --> task score (-): 13.0, task score (+): 21.0, contribution: 8.0
Robot 2, Task 1 --> task score (-): 18.0, task score (+): 18.0, contribution: 0.0
Robot 3, Task 0 --> task score (-): 17.0, task score (+): 21.0, contribution: 4.0
Robot 3, Task 1 --> task score (-): 18.0, task score (+): 18.0, contribution: 0.0

Robot 0, Task 1 --> 0 gain -2.0
Robot 0, Task 1 --> 1 gain 0.0
Robot 0 proposes stay @ 1 (gain: 0.0)

Robot 1, Task 1 --> 0 gain -5.0
Robot 1, Task 1 --> 1 gain 0.0
Robot 1 proposes stay @ 1 (gain: 0.0)

Robot 2, Task 0 --> 0 gain 0.0
Robot 2, Task 0 --> 1 gain -8.0
Robot 2 proposes stay @ 0 (gain: 0.0)

Robot 3, Task 0 --> 0 gain 0.0
Robot 3, Task 0 --> 1 gain -4.0
Robot 3 proposes stay @ 0 (gain: 0.0)
# <-- end of round 3 -->

------------------------------------------------
Round 3, time: 0.018 seconds, total utility: 39.0
------------------------------------------------

Allocation finished in 3 rounds, time: 0.018 seconds, total utility: 39.0

# final allocation
# --------------------
  Unallocated > []
       Task 0 > [2, 3]
       Task 1 > [1, 0]
# --------------------
```

Listing 1. Task allocation example with 2 tasks and 4 robots.

TASK RECOGNITION AND OPTIMAL SEQUENCING

The Task Recognition and Optimal Sequencing module serves a dual purpose within the context of the Symbiotic Orchestration Module. On the one hand, it is responsible for identifying opportunities where smart agents can collaborate with, and support, each other. This functionality is especially useful in situations where one smart agent faces some difficulty in accomplishing its current task, such as being stuck because its path is blocked by an

object not accounted for during the path generation process. On the other hand, this component is responsible for optimizing the command sequences given to the agents, so that specific system-wide goals are maximized. An example of such a goal is to increase the overall time in operation across all smart agents, by conserving as much energy as possible.

In order to be able to accomplish its objective, the Task Recognition sub-module takes into account important parameters of ongoing missions, such as the agent execution instructions, as well as the status messages and telemetry data from the agents involved in these missions, and determines if any of its predefined rules for multi-agent collaboration are satisfied by the current conditions. The component follows an event-based approach for accomplishing the task recognition objective, so that the appropriate checks for collaboration opportunities are not evaluated constantly, but only after events that have high probability of triggering such opportunities. Such an example would be when an agent is low on battery or is blocked by an obstacle. However, if more frequent evaluation is needed, then time-based events can also be employed for this purpose. In this case, a timer configured by the system to run every few seconds can produce an appropriate event so that the Task Recognition sub-module is also triggered at the same interval.

When the predefined set of requirements for a collaboration opportunity are satisfied, the component proposes to the system a symbiotic task that satisfies the needs of the particular scenario. From that point onward, higher-level modules or the first responders at the command and control center can choose to accept or decline the proposed symbiotic mission.

The Optimal Sequencing sub-module is triggered when new symbiotic missions are proposed by the system. It takes as input the agent execution instructions and checks if any of its predefined rules for a specific objective can be applied to these execution instructions in order to fulfill the objective. For instance, if a UAV and a UGV are both instructed to go from point A to point B and the system-wide objective for the Optimal Sequencing module has been set to maximize the time in operation for the agents, i.e. conserve battery when possible, then this module can introduce a delay in the initial *Goto* command given to the UAV, so that it remains landed at point A while the slower UGV starts executing the same command. After this delay, the UAV is also instructed by SOCM to execute the *Goto* command so that the two agents arrive at point B at the same time. The Optimal Sequencing sub-module is also responsible for appropriately sequencing the commands given to agents in symbiotic missions in order to avoid blocking interactions among them, such as the UAV being instructed to enter a room before the UGV has opened the door for that room.

CONCLUSION

This paper presented the components that constitute the Symbiotic Orchestration Module, the main task of which is to facilitate the collaboration of smart agents (UAVs, UGVs) in disaster response scenarios. By coordinating the actions of different agents, which may have varied and sometimes complementary capabilities, this module is able to enhance the overall capabilities of the system beyond what is possible by individual agents that act in isolation. The first such component was the Mission Controller, which monitors all proposed missions and handles the execution of single-agent missions. Then, the Symbiotic Operation Control Module was presented, which handles the execution of multi-agent collaborative missions. These modules communicate with the agents in a commonly understood "language", the mission execution instruction set, which can be expanded and adjusted in order to cover additional capabilities and agents. The next component that was presented, the Task Allocation Module, automates the allocation of agents to missions in cases where the mission instructions do not specify a particular agent. Finally, the Task Recognition and Optimal Sequencing Module, is responsible for identifying opportunities for agent collaboration and for system-wide goal optimization.

ACKNOWLEDGMENT

This work was supported by the European Project: INTREPID Grant no. 883345 within the H2020 Research and Innovation Programme.

REFERENCES

Draganjac, I., Miklić, D., Kovačić, Z., Vasiljević, G., and Bogdan, S. (2016). “Decentralized control of multi-AGV systems in autonomous warehousing applications”. In: *IEEE Transactions on Automation Science and Engineering* 13.4, pp. 1433–1447.

Ebel, H., Luo, W., Yu, F., Tang, Q., and Eberhard, P. (2020). “Design and experimental validation of a distributed cooperative transportation scheme”. In: *IEEE Transactions on Automation Science and Engineering*.

Fitzpatrick, S. and Meertens, L. (2003). "Distributed Coordination through Anarchic Optimization". In: *Distributed Sensor Networks*, pp. 257–295.

Guruprasad, K. and Ghose, D. (2010). "Automated multi-agent search using centroidal voronoi configuration". In: *IEEE Transactions on Automation Science and Engineering* 8.2, pp. 420–423.

Hichri, B., Adouane, L., Fauroux, J.-C., Mezouar, Y., and Doroftei, I. (2016). "Cooperative mobile robot control architecture for lifting and transportation of any shape payload". In: *Distributed Autonomous Robotic Systems*. Springer, pp. 177–191.

Hichri, B., Fauroux, J.-C., Adouane, L., Doroftei, I., and Mezouar, Y. (2019). "Design of cooperative mobile robots for co-manipulation and transportation tasks". In: *Robotics and computer-integrated manufacturing* 57, pp. 412–421.

Kernbach, S., Meister, E., Schlachter, F., Jebens, K., Szymanski, M., Liedke, J., Laneri, D., Winkler, L., Schmickl, T., Thenius, R., et al. (2008). "Symbiotic robot organisms: REPLICATOR and SYMBRION projects". In: *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pp. 62–69.

Lee, H., Kim, H., and Kim, H. J. (2016). "Planning and control for collision-free cooperative aerial transportation". In: *IEEE Transactions on Automation Science and Engineering* 15.1, pp. 189–201.

Li, Q., Li, M., Vo, B. Q., and Kowalczyk, R. (2020). "Distributed Near-optimal Multi-robots Coordination in Heterogeneous Task Allocation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4309–4314.

Macarthur, K., Stranders, R., Ramchurn, S., and Jennings, N. (2011). "A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, W., Wang, G., and Wittenburg, L. (2002). "Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance". In: *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*.

Zhang, W., Wang, G., Xing, Z., and Wittenburg, L. (2005). "Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks". In: *Artificial Intelligence* 161.1–2, pp. 55–87.