Proposal of a Mission Management Module for Embedded Agent Systems

Georgiy C. Tanca Nazarov[1], Iago Silvestre[1], Fernando R. Santos[1], Jomi F. Hübner[1], Leandro Buss Becker[1]

[1]Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brazil

{georgiytn, iagosilvestre2004, fernando.rod.santos}@gmail.com,

{jomi.hubner, leandro.becker}@ufsc.br

Abstract. *Multiagent Systems have characteristics suitable to be used in the control of Multi-Robot Systems (MRS), but due to the different missions that can be performed in an application, and the complex interactions among them, some difficulties are observed in their implementation. Therefore, we propose a Mission Management Module (MMM), to be used in Embedded Agent Systems, that is capable of switching between missions as required, while keeping track of mission progress, preventing task repetition when returning to a previously interrupted mission. To design this proposal, we define key concepts and requirements to assist and delimit the scope of this module. In addition, we propose a general architecture of a system, within which the MMM is encapsulated, to be used in MRS applications. An initial implementation of the MMM in Jason Language is presented. Finally, we assess this initial implementation and suggest further improvements in order to comply with the proposed requirements.*

1. Introduction

In the application of Multi-Robot Systems (MRS), the control of Unmanned Aerial Vehicles (UAVs) is a field of high relevance due to the variety of uses that this type of embedded system has, from civilian, educational, commercial and even for modern military applications. Moreover there is great interest in the implementation of autonomous control in UAVs, enabling them to complete tasks with minimal human intervention, sometimes due to a shortage of skilled human operators or due to the repetitiveness of certain operations [Silvestre et al. 2023]. There are promising advancements in the field MRS that enable the cooperation of individual elements of a system in order to enable better results for certain applications [Verma and Ranga 2021].

Multiagent Systems (MAS) and the Agent-Oriented Programming (AOP) paradigm provide features such as autonomy, coordination and scalability for applications [Olivier et al. 2020]. Thus some researchers are making use of the MAS approach to fully or partially control single and multiple embedded systems.

A current line of research is to determine a way to switch between the different missions that a UAV application can have, which we denominate as **mission management**. Each individual UAV can have multiple missions that can be switched based on changes in either the external environment or its internal state. As an example, a UAV has a set of coordinates that it needs to visit as part of its mission, but in the midst of it, the

system sends an alert that the battery level is low, so for its safety the system switches to the mission to return to the charging station. Once it finishes charging, the UAV should resume its previous mission from the last point it reached. It might appear trivial to implement the switching as a simple condition trigger while programming each mission, however this approach does not scale well as the number of missions grows.

In order to simplify and streamline the creation of new missions and the deployment of new applications, we propose a Mission Management Module (MMM) that is capable of switching automatically between different missions. This management module is part of a larger system architecture that we propose for an Embedded Agent System (EAS), with the goal of realizing an on-board autonomous control system that could be implemented independently of the target application.

The remainder of this paper is divided into five sections; in Section 2, we detail the problem and define requirements and key concepts used to idealize the functions of the MMM. In Section 3, we present related works that tackled similar problems. The MMM is integrated within a proposed architecture for the EAS that is discussed in Section 4. In Section 5, we present an initial implementation of the module using the agent programming language, Jason. Finally, in Section 6 we present our conclusions and, since this is a work in progress, we discuss further improvements of the initial implementation.

2. Problem Overview

The initial motivation to implement the Mission Management Module is the participation on the 2022 SARC–BARINET Aerospace Competition. The theme of the competition was fire detection and firefighting in forests using UAVs. A simulation suite was provided and the goal of each participating team was to design and implement a mission control system to search and extinguish the fire while meeting some specific requirements by which the team's score was assessed. The proposal employed a MAS approach to design and implement an autonomous control system for multiple UAVs using Jason, an AOP language [Silvestre et al. 2023].

One of the main challenges of the competition was the effective use of collaboration strategies between several UAVs to reach a common goal. The MAS approach enables this feature as it is part of its core functionalities [Olivier et al. 2020]. Even though the scope of the competition was purely simulation based, one of the main qualities of the implemented system was decentralization and distribution. The system proposed did not have a central point of control, instead each vehicle was controlled by an embedded agent that was able to perform its missions alone and cooperate with other agents (UAVs) when needed.

This system was further developed towards realizing it on a real-life scenario [Silvestre et al. 2025]. For that, new requirements appeared that were not contemplated on the simulation, such as dealing with the finite nature of resources the UAV has in order to complete its missions, e.g. battery-power and fire retardant levels. This, coupled with the cooperative nature of the solution, meant that each embedded agent had to switch between multiple missions as part of a reaction to its internal state, environment, and other agents around it. In addition, in order to avoid task repetition when switching back to an on-going mission, the system needed to keep track of the tasks already completed.

The initial approach to face these challenges was by programming conditions that

needed to be checked continuously in order to stay in the same mission or to switch to another one if so required. However, the number of different interchangeable missions grew and so did the need to individually program trigger conditions, which became burdensome as extensive parts of the original code had to be constantly checked and reworked any time a new mission was added or an existing one was modified. This exposed the need of creating a mission management algorithm, capable of dealing with these issues. Table 1 compiles the requirements that it needs to fulfil, some of them are an extension of the requirements from past works, and classifies them into functional and non-functional.

Table 1. Mission Management Algorithm Requirements

Functional Requirements		**Non-Functional Requirements**	
F1.	Reinforce autonomy	NF1.	Application independent
F2.	Switch missions as required	NF2.	Capable of prioritization
F3.	Avoid task repetition	NF3.	Keeps a backlog of missions
F4.	Mission abstraction	NF4.	Compatible with planning algorithms
F5.	Self-contained	NF5.	Optimized for embedding

2.1. Definition of Concepts

It is useful to conceptually define some frequently used terms. We define a **mission** as a set of tasks, a **task** as a set of actions, and **actions** as the basic operations the system can perform. The key difference between a mission and a task is the overall level of abstraction we want to grant to the system, so in general it is up to the designer to decide what constitutes a task and what would constitute a mission depending on the application.

Taking the example of the competition, searching for fire is defined as a mission, since it has various steps at which it could be interrupted. This mission could be defined as a set of coordinates that the UAV needs to travel to and actively search for fire, and the individual tasks are to go to each coordinate and search for fire. Fighting the fire would be a separate mission, since it has a specific goal and has its own tasks to complete. More importantly, as we divide these missions into tasks, it allows the system to have a point at which it could return to in case the mission was interrupted, thus avoiding a complete restart of the mission. This means that missions should be composed of tasks granular enough to, in case of interruptions, avoid too much task repetition.

2.2. Refinement of Concepts

We established some constraints to refine our initial concept of mission and task, with the aim of reducing the complexity and improving feasibility, when designing and implementing the system.

- The system must have knowledge of when a task was completed, which means that it should receive feedback informing if the task was a success or a failure.
- The completion of a task must be deterministic, so there is no uncertainty involved in the given feedback.
- A mission must be an ordinal set or a sequence of task, this means that each task has an order at which it should be completed.

As the concepts of mission and task are similar, we also define that the two properties of a task are also applied to missions. This means that when a mission is completed it should receive deterministic feedback about it. The purpose of a mission is to divide a greater goal into its basic steps, crucially it is in between these steps that a mission could potentially be interrupted, and later on resumed.

2.3. Mission Management

We propose a *mission management*, that uses these definitions discussed so far and complies with the requirements of Table 1. Its main function is to coordinate the switching between the different missions of the system, given the current context at which it operates. For that, this module must have a knowledge base including all possible missions, this also means that it has to know the individual tasks of each mission. It should be capable of receiving and interpreting the feedback signal of each task, and keep temporary knowledge of all the active missions and their completed tasks. By defining some specific properties, the mission management will be able to assess whether it should interrupt the current mission, or rather to add the new one to a list of suspended missions, or even to ignore it.

These specific properties are a useful tool to imbue aspects of mission flow control to the user, as it is with them that the behaviour of the autonomous system can be controlled. A very useful property is one that gives the notion of priority, as in many applications there will always be a need to prioritize certain goals over others, whether because of safety concerns or even for optimization purposes. A simple way solve this is to give an ordinal property to each mission. Another property that could be used is the ability to allocate resources, this could potentially be used to implement the notion of concurrency between missions, meaning that multiple missions that use different available resources could be running in parallel. It is key to emphasize that these properties are useful tools we use to enrich and improve the decision making capabilities of the mission management, but are not strictly necessary for its most basic function.

The mission management is not concerned with how the missions are created, but only with how the missions interact with each other during deployment. The role of creating the mission, defining its tasks and its properties, could be done by a human operator, but alternatively it could be produced by another method. We denominate such method as *mission planning*. Clearly separating the roles of managing the missions from the one of planning the missions is essential. This is because, while the mission switching problem is fairly straight forward and mostly immutable, the problem of mission planning is varied and highly dependent of the specific application of the entire system.

3. Related Works

The design of embedded autonomous systems for usage in autonomous vehicles is a problem that has been extensively researched throughout this century, be it for terrestrial, aerial or aquatic applications. In this section, we discuss some works related to this, in order to get an overview of system architectures and then compare them with our proposed architecture. Although our inspiration was a problem proposed for the specific use case with UAVs, our selection includes examples from other types of vehicles, since the intention of our proposed architecture is to be as comprehensive as possible so that it can be used in most embedded systems.

On [Pebody 2007] a control system architecture for use in Autonomous Underwater Vehicles (AUV) is presented. The system architecture consists of a layered approach with three control layers, the lower Control Behaviour Layer is in charge of elementary control mostly consisting of simple closed loop processes of actuators and sensors; the middle Mission Sequencing Layer has the role of issuing commands in a timely manner to reach a certain behaviour on the vehicle; and the Mission Planning Layer has the role of providing the actual commands, and was initially delegated to a human operator, but later an autonomous planning script is proposed. This work has a discussion about some of the concepts mentioned before as mission abstraction and granularity of tasks and how it affects the mission planning stage. Missions are represented as a sequence of commands given to the vehicle to achieve a particular goal. Finally, it debates how these concepts affect mission reliability and success rate. In this case all mission planning is done before deployment, and once the mission is running it is implied that only minimal alterations occur, i.e. minimal reaction. The top layer of mission control could be implemented in an external computer or on-board, while the two lower layers exists solely on-board.

In the works of [Pascarella et al. 2013a, Pascarella et al. 2013b] a system architecture to implement an autonomous control system of a UAV was proposed. It consists of a mission planning algorithm implemented using Belief-Desire-Intention (BDI) Agents to solve the trajectory planning problem within given constraints and real-time monitoring for re-planning. The mission planner would generate a sequence of waypoints as part of a mission that the flight controller would have to visit. But unlike the previous work, the implementation of the mission planning algorithm, was only made in simulations and with hardware-in-the-loop tests, i.e. without using a real UAV. Although this work was mostly focused in the mission planning aspect of the system, it stated the benefits of implementing the mission control agent system on an on-board computer.

A platform for control of multiple UAVs is presented in [Temme et al. 2022]. The traffic and mission management component is the main feature of this platform, it has the goal of coordinating and scheduling missions on multiple UAVs. It has a data pool of selected missions that a human operator can request and the system will automatically plan the flight trajectory and assign an available UAV to complete it. During execution the system supervises the UAV and if the need arises resolves conflicting trajectories between different UAVs and updates flight trajectories for them. While the mission planning and conflict resolution is done automatically, each individual UAV still requires a human operator to perform critical take-off and landing operations, which limits the autonomy of the whole system. Moreover, the centralized nature of the system could be highlighted as another limitation, as all of the high level mission control is made around a centralized component which requires constant connectivity with the vehicles to react to and resolve conflicts.

The work of [Silva et al. 2023] introduces a mission planning framework to control multiple UAVs, it uses user defined adjustments and constraints and modifies missions in real-time according to events. The paper takes into consideration the benefits of having a decentralized approach for such a framework but ultimately relegates the mission planning to a centralized system outside the UAV. It also features a mission management component, internal to the planning framework, that keeps track of the status of current missions, and reacts to faults detected during runtime to cancel missions. Another char-

acteristic of their architecture, while not explicitly mentioning it, is the delegation of the most basic control of the UAV to a flight controller, while the framework dealt with a more abstract mission plan. The framework was tested both in simulations and in real use cases with flying UAVs.

As we have seen all these works related to embedded autonomous systems have used the notion of mission, but so far only [Pebody 2007] has discussed the need of formal definition of what constitutes a mission and even the issue of granularity of the tasks that constitute it. Most of them stated the benefits of using a more abstract language to represent the mission and its tasks, except [Pascarella et al. 2013a]. All of them had designed systems bound to their particular application, except [Silva et al. 2023] which discussed the similarities that exist between autonomous embedded systems and foresaw the possibility of using the same software in different applications with minimal adjustments. And in every work, the systems react to events, albeit with varying degrees of autonomy and mission modification. Both [Pascarella et al. 2013a] and [Silva et al. 2023] have mentioned the benefits of using a decentralized approach, nevertheless, only the former actually implemented the system on an on-board platform.

A noteworthy characteristic is that all of them consider the functions that we attribute to the mission management either as an intrinsic part of the mission planning stage or as an element encompassed by the mission planning algorithm. This clearly differentiates from our proposal, that clearly separates these functions in distinct modules.

4. Proposed System Architecture

Here we present the general agent architecture that the proposed MMM is part of. This architecture is based on the model described in [Santos et al. 2015] and possesses two levels of abstraction: Planning Level and Control Level, as shown in Figure 1. This allows decoupling the hardware from the application, as each is controlled in a different level and they coordinate the system behaviour with the use of a common interface.

The **Control Level** pertains to the low-level control, and interacts directly with the specific hardware of the system, with three modules: one for monitoring the sensors, one for driving the actuators and one for controlling the system dynamic. This level is outside the scope of this work, so no further details on its functionality are provided.

The **Planning Level** pertains to the high-level control and has three modules: Mission Planning, Mission Management and Deliberation. This level is responsible for deciding the operations the system is currently executing and also those it is planning to execute later.

The connection between these levels is made through an *Interface*, capable of bridging the communication between them. Furthermore, a modularized approach to the Planning Level offers the benefits of a decentralized and distributed system [Coulouris et al. 2011]. Thus, it allows the embedded system to have modules running on different machines, eliminating the vulnerability of a single point of failure that could compromise the entire system.

Furthermore, the Deliberation module represents the internal algorithm with which the embedded agent processes its beliefs, goals and plans. This will generally depend on the type of agent framework used, and so the design of this module is not

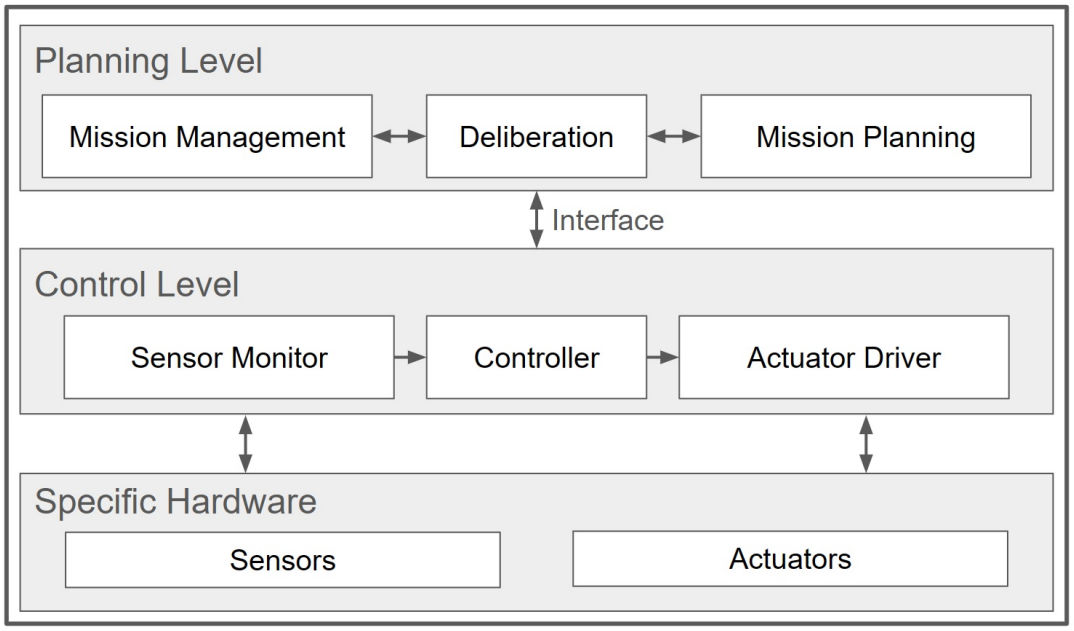

Figure 1. General Architecture for the Embedded Agent System

the focus of this work. Although, the architecture does anticipate that this deliberation process will make use of the Mission Management and the Mission Planning modules.

As we explained in Section 2.3, the main function of the **Mission Management Module (MMM)** is to orchestrate the execution of missions and tasks of the system. As for the **Mission Planning Module (MPM)**, its main function is to create missions and specify the tasks that must be completed to achieve them. A human operator could fulfil this role and create the missions before deployment as needed, but it could also be done with the use of planning algorithms [Pascarella et al. 2013a], though this falls outside the purview of this paper.

A question that arises from the interaction between the MMM and MPM is: Why isn't the MPM the only one controlling the autonomous behaviour of the system? In a perfect world, the MPM would be omniscient and thus it would not need the MMM, as it would know exactly when each mission should be executed and would know beforehand any problem and then plan around them. But in the real world uncertain conditions arise, and so there is a need for the system to be able to react to them in real-time and return to its normal mission whenever it is possible.

Another reason to separate these modules is that planning algorithms are very costly to use in regards to computational time and energy required, which makes it incompatible with some embedded systems [Lee and Seshia 2017]. This is why in many cases the mission planning is done offline, or on a remote ground control station. Our goal with this architecture is for it to be mostly embedded within the vehicle, so it is beneficial to use the simpler MMM that is capable of switching between missions already specified. While the costly MPM *could* be executed in a remote station and periodically communicate with the vehicle.

Our architecture does not specify a clear hierarchy or dependency between the

MMM and MPM. This is because the planning module does not need to interact directly with the management module. These modules are tools for the embedded agent to use, ultimately it is up to the agent to decide how to react to a given event, and it could very well use either or both of them depending on the situation, adding flexibility to the level of autonomy we want to imbue the system.

5. Implementation of the Mission Management Module

We developed an initial implementation of the proposed MMM, despite some limitations from our originally idealized architecture. This was the result of an extension of the work described in [Silvestre et al. 2023]. In this initial work, the system architecture is divided in levels, with Jason agents dedicated to the higher-level mission control of individual UAVs, and a bottom level implemented via Robot Operating System (ROS), which was in charge of handling the low-level control procedures. It demonstrated the benefits of using an agent based approach, as it allowed to attain a higher level of programming code abstraction when compared to traditional imperative languages. It also allowed easier scalability of the solution, as Jason agents support cooperation of distributed and decentralised teams. Lastly, it possessed hardware independence, as the modularization and abstraction of the system meant that high level control commands could be interpreted by the level below it with the implementation of an interface, in this case a Jason-ROS interface.

As described in Section 2, the addition of more and more missions incentivized the creation of a Mission Management Module on the system. The implementation of the MMM was done as part of a library that could extend the original functionality of any Jason agent, with its own set of plans intended to fulfil the functional requirements outlined in Table 1. The main function of this library is to orchestrate the execution of the different missions that arise during deployment, and streamline the process of adding new missions to the application.

The library defines the *attributes* required for each mission as follows: a unique **identification**; its **type**, this can be *atomic* (for missions that can never be interrupted), *resumable* (for missions that can be suspended and resumed later), and *droppable* (for missions that can be stopped and abandoned); a **list of tasks** to accomplish the mission; and its **current state**, which could be *created*, *running*, *suspended*, *dropped* or *finished*. In the life-cycle of a mission, first it is *created*, once the command to execute it is given, its state is *running*, and once the final task is completed, its state becomes *finished*. If a mission is interrupted, depending on its type, its state can either become *suspended* or *dropped*, as shown in Figure 2.

By using the mission attributes, the library allows the agent to decide to suspend an on-going mission (if applicable), and switch to a different one. After completing this last mission, it automatically resumes the suspended mission, exactly after the last task that was completed in it. This eases the burden of updating plans of an already existing mission whenever a new mission is added, as the programmer only needs to properly design the attributes of the new mission and the triggering condition to run it in order to know how the mission will affect the behaviour of the system during runtime. This implementation foregoes the use of a MPM for now, instead all the missions and their tasks are planned before deployment by the programmer. In spite of that, the overall a

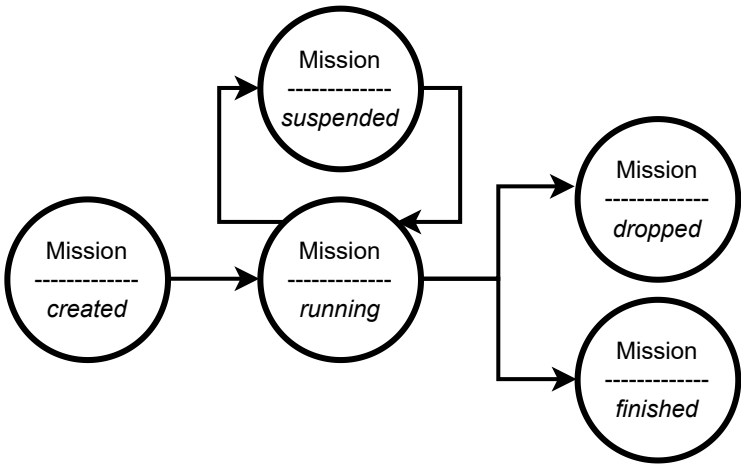

Figure 2. Mission Life Cycle States

system using the MMM can still display a high level of autonomy, since the agent can dynamically react to the environment it perceives and switch missions accordingly with the help of the MMM.

5.1. Use Case Example

Although the initial implementation was used to manage drones in a firefight scenario, a simple "*Hello World!*" type application is used here to demonstrate the functionalities of the MMM and the plans the Jason agent is required to have to properly use the module. This example is focused on the use of the MMM and so it does not communicate with a lower level of control. This application's purpose is to execute a *resumable* mission that prints on the console the phrase *Hello World!*, one word at a time, but along its execution an *atomic* mission can interrupt it, and this other mission prints a different message on the console. In this example the Deliberation module is the Jason agent and, in order to complete its mission, it executes a command to print a specific string on the console, which in the example would be our Interface. Figure 3 depicts a step by step interaction that the Deliberation Module (*Jason agent*) has with the MMM and with the Interface (*Console*).

To use the MMM, the agent first needs to create the mission, this is done by executing the command `create_mission(Id,Type,Tasks)` with the next arguments: a unique identification, its type and the sequence of tasks. Once the mission is created the agent can execute `run_mission(Id)` with the unique name of the mission. When that command is issued, then the MMM checks if there is a mission currently running, if not, it adds the goal `run_plan(Id,Tasks)` to the agent. This means that the agent must have a plan programmed to complete the task. And in case there is a mission currently executing, the MMM checks if this current mission is atomic or not; if it is, then the new mission is suspended and the current mission continues its execution; if it is not, then the current mission is suspended and the new mission is executed instead.

Whenever a task is completed the agent must give a feedback to the MMM, via the signal `update_mission(N)`, where N is the number of the task that were completed so far. Once N matches the number of total task for the current mission, the MMM interprets that the mission was finished and changes its status. The MMM then resumes a

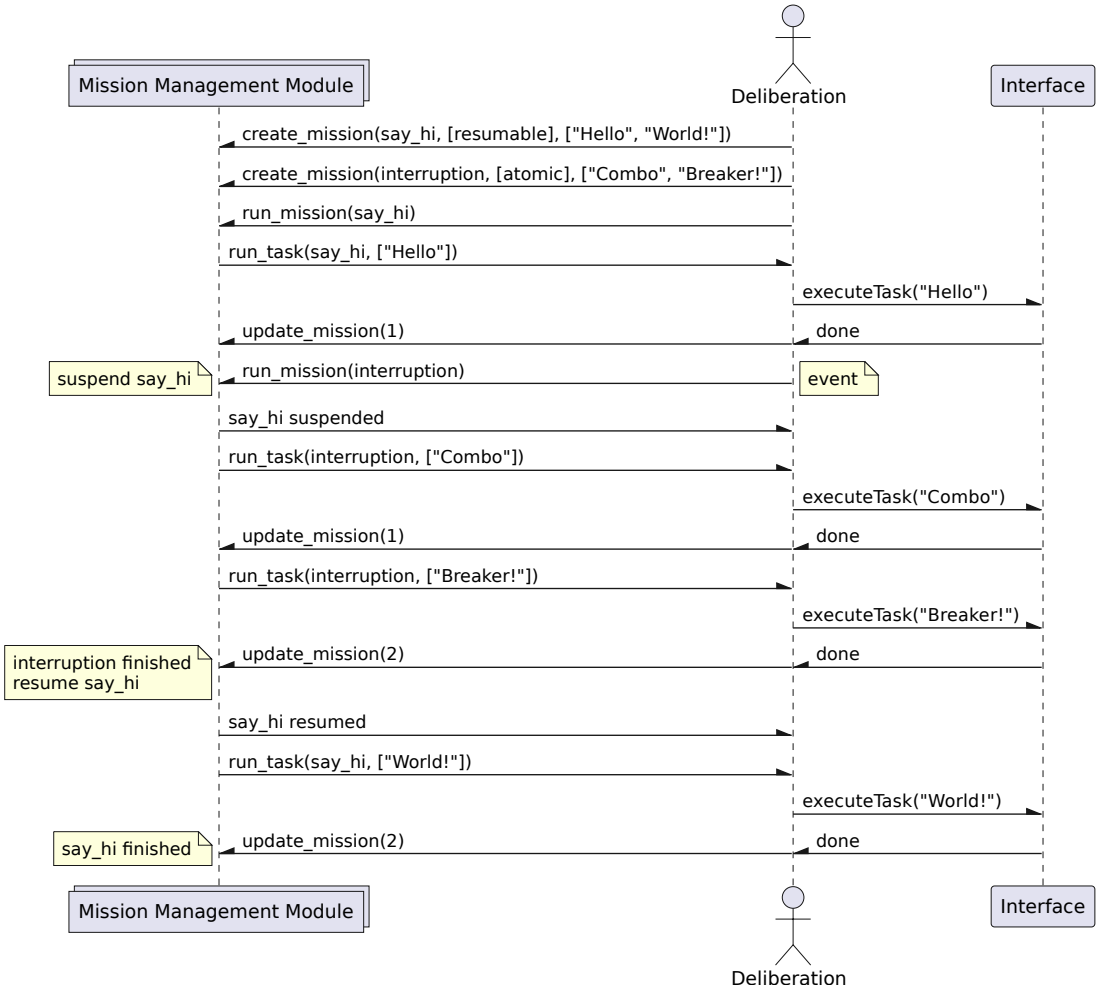

Figure 3. Interaction between the modules in the *Hello World!* example

suspended mission, if one exists.

```
1  +!my_missions
2      <-  !mm::create_mission(say_hi, [resumable], ["Hello","World!"]);
3          !mm::create_mission(interrupt, [atomic], ["Combo","Breaker!"]).
4
5  +!start <- !mm::run_mission(say_hi).
6  +event  <- !mm::run_mission(interrupt).
7
8  +!mm::run_task(_,Text) <- .print(Text).
```

Listing 1. Code snippet of Hello World example.

A snippet of the agent code for this example can be seen in Listing 1, containing the agent's plan that creates the missions. We also see the trigger to run the first mission of the example, and below the plan that will be triggered to complete a task from it. The complete code of this application is available through our GitHub repository. [1]

[1] https://github.com/jason-lang/mission-manager

6. Conclusion

We proposed a MMM for use in EAS, and outlined the requirements that this module must comply with. As part of this proposal we defined the concepts of mission, task and action. Complementing this, we established constraints to these concepts to reduce complexity and improve the viability of the module. Along with this, a general architecture to use in Embedded Systems was proposed, that makes use of the MMM. This architecture and module are independent of any specific technology for embedded systems, in order to be used in a broad type of applications.

An initial implementation of the MMM was presented, and was able to satisfy the functional requirements we proposed. However further testing and improvements are necessary in order to properly satisfy the non-functional requirements. We have identified some deficiencies that MMM possesses, and plan to address them in future works. One of these is that, while we consider that the MMM is self-contained, the number of interactions that the agent must have with it could be reduced. By reducing the number of lines of code that the programmer needs to add to create a new mission, it would also improve the mission abstraction aspect too.

Another problem to address is the way the MMM handles the cases where more than one mission is suspended, currently the MMM by default resumes the most recent suspended mission. In a hypothetical scenario where, during the execution of mission A, the command to execute mission B is issued, means that mission A is suspended. During execution of mission B, a command to execute mission C is issued, so mission B is also suspended. When mission C finishes, the MMM will resume mission B, as it is the last suspended mission. We consider this restrictive in terms of mission flow control.

We categorize the missions as atomic, resumable and droppable, in order to give a sense of importance of each mission, but this type prioritization is fairly limited. Instead of using these categories, we believe that a more explicit way of assigning priority, that is with cardinals, will grant a more transparent way of evaluating priorities, both for the MMM and for the mission designer. Nonetheless, this categorization could still prove useful and could be reused as additional mission properties.

One way we can address this and the previous issue mentioned, is by creating a queue of missions to be executed. And by improving the prioritization logic, we can add more flexibility as to how the missions are added to this queue. So in the case that there are more than one suspended mission the MMM will resume missions according to their priority level.

Usually an UAV is designed to complete a single specific task that it was designed to, however, another kind of mobile vehicle could perform multiple different tasks at the same time. As our proposed architecture has the objective of being application independent, we consider that adding the possibility of having concurrent missions is highly beneficial. To implement such capability we could mimic solutions that are already being used in concurrent computing.

All the testing done to the MMM was done while working exclusively in the Planning Level or by using a Control Level coupled with simulations as a substitute for the embedded hardware. Future works include doing hardware-in-the-loop tests to assess if it is feasible to use the proposed Mission Management Module in an embedded system,

and to determine the computational capabilities that such system should have. Furthermore, the implementation of real applications that make use of the proposed architecture is planned.

References

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison Wesley Longman.

Lee, E. A. and Seshia, S. A. (2017). *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. MIT Press.

Olivier, B., Bordini, R. H., Hübner, J. F., and Ricci, A. (2020). *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*. The MIT Press.

Pascarella, D., Venticinque, S., and Aversa, R. (2013a). Agent-based design for uav mission planning. In *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 76–83.

Pascarella, D., Venticinque, S., and Aversa, R. (2013b). Autonomic agents for real time uav mission planning. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pages 410–415.

Pebody, M. (2007). The contribution of scripted command sequences and low level control behaviours to autonomous underwater vehicle control systems and their impact on reliability and mission success. In *OCEANS 2007 - Europe*, pages 1–5.

Santos, F. R., Hubner, J. F., and Becker, L. B. (2015). Concepção e análise de um modelo de agente bdi voltado para o planejamento de rota em um vant. *Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)*.

Silva, M., Reis, A., and Sargento, S. (2023). A Mission Planning Framework for Fleets of Connected UAVs. *Journal of Intelligent & Robotic Systems*, 108(1):2.

Silvestre, I., Becker, L. B., Fisher, M., Hübner, J. F., and de Brito, M. (2025). Enhanced agent-oriented programming for robot teams. *Engineering Applications of Artificial Intelligence*, 158:111390.

Silvestre, I., de Lima, B., Dias, P. H., Buss Becker, L., Hübner, J. F., and de Brito, M. (2023). Uav swarm control and coordination using jason bdi agents on top of ros. In Mathieu, P., Dignum, F., Novais, P., and De la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*, pages 225–236, Cham. Springer Nature Switzerland.

Temme, A., Kuenz, A., Lieb, T. J., and Friedrich, M. (2022). Traffic and mission management in the respondrone project. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–6.

Verma, J. K. and Ranga, V. (2021). Multi-robot coordination analysis, taxonomy, challenges and future scope. In *Journal of Intelligent & Robotic Systems*. Springer.