The Mysterio framework for developing cooperative Multi-UAV Systems

Antônio Sávio Nascimento Cavalcante
Universidade Estadual de Campinas
Campinas, São Paulo, Brazil
savionasc@gmail.com

Breno Bernard Nicolau de Franca
Universidade Estadual de Campinas
Campinas, São Paulo, Brazil
bfranca@unicamp.br

ABSTRACT

Over the years, UAVs (also known as drones) have been growing in studies and applications to solve diverse problems. Due to the complexity of these problems, dealing with just one UAV may not be enough, but using several UAVs together to work cooperatively increases its capacities, thus boosting solutions. However, developing cooperative Multi-UAV systems is not trivial, and reuse support is usually limited to low-level implementation. This work presents a framework for Multi-UAVs, called Mysterio, which provides an underlying software architecture with essential Multi-UAV components, enabling the reuse of design and code so that engineers can instantiate it to carry out specific missions by making UAVs work in cooperation. We also present four instances of the framework to evaluate Mysterio's effectiveness in the face of the developed scenarios. Finally, we discuss the framework's potential to provide and support reuse code to develop Cooperative Multi-UAVs systems for different application scenarios.

CCS CONCEPTS

• **Software and its engineering** → **Software architectures**; **Reusability**; • **Computing methodologies** → *Simulation evaluation*; • **Computer systems organization** → **External interfaces for robotics**.

KEYWORDS

MultiUAV Systems, Framework

ACM Reference Format:
Antônio Sávio Nascimento Cavalcante and Breno Bernard Nicolau de Franca. 2022. The Mysterio framework for developing cooperative Multi-UAV Systems. In *16th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS 2022), October 3–4, 2022, Uberlandia, Brazil*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3559712.3559718

1 INTRODUCTION

Research on Unmanned Aerial Vehicles (UAVs) has emerged over the years and it is becoming popular in both civil and military applications [5, 11]. There are several examples of UAV applications for carrying out operations in different fields such as agriculture [32], risky operations such as fire management [14], search and rescue [25] and others. Furthermore, the use of UAVs presents several technical challenges like planning algorithms [24, 29], area coverage and mapping [35], integration with Internet of Things solutions [19], among others.

Pursuing efficiency in the use of UAVs, it is possible to maximize the advantages through cooperation in a network setting of multiple UAVs, also called Multi-UAV systems. Although dealing with multiple UAVs increases the complexity of control and communication, there are also advantages. For instance, Multi-UAV systems allow redundancy, increasing scalability, reliability, availability, and also survivability [5, 11]. Another advantage is the reduction of overall execution time, given the number of nodes executing tasks [26].

Building cooperative Multi-UAV systems requires a communication architecture, which should be enough to not interfere with their cooperation, even if some UAVs are not available at all times. The design of Multi-UAV systems foresees some topologies that impact architectural decisions in a Multi-UAV system [5]. Structuring a system in a specific topology may perform better than providing support to multiple topologies. For example, all the UAVs communicating directly with a base station may be enough to achieve the system goals. However, depending on the scenario, letting some UAVs communicate directly with each other is also a feasible alternative.

The idea of cooperative Multi-UAV systems is not new. In [33], the authors developed cooperative search strategies to find moving or stationary targets. The communication between drones was fundamental in cooperative missions, whether in the target identification algorithm or the reorganization of the UAVs in case of failures.

In addition to topology and cooperation, a well-designed software architecture represents a key success factor in the performance of UAVs tasks and the cooperative system behavior as a whole [6, 28]. The selection of technologies for the development of such an architecture is essential, that is, technologies such as hardware, frameworks, and protocols impact several attributes of the system.

The cost of designing these systems can be critical to development, so open-source and reuse-oriented technologies can facilitate development by simplifying the reuse of architecture design and code, and saving development effort [27].

To achieve the expected benefits of UAVs working cooperatively, an architecture needs to consider issues such as design cost, topology changes, mobility, and energy constraints, among others [1, 11].

SBCARS 2022, October 3–4, 2022, Uberlandia, Brazil

ACM ISBN 978-1-4503-9745-2/22/10…$15.00
https://doi.org/10.1145/3559712.3559718

These and other challenges make it difficult to develop a high-quality architecture promoting communication and cooperation in Multi-UAV systems.

With the combination of UAVs into Multi-UAV systems to make them cooperate to achieve their goals, the complexity of such systems increases, as well as the need to organize and modularize their software components, targeting the functional correctness and flexible evolution. This set of needs makes the development of Multi-UAV software a complex task. Still, the lack of technologies focused on design and code reuse for the development of this type of system makes it challenging, in addition to the challenge of organizing them architecturally to accomplish their tasks efficiently, whether individual or collective tasks. Thus, a poorly designed architecture in Multi-UAV systems makes them inefficient and does not exploit their expected benefits, such as the short time of operation in carrying out a mission.

As seen in several works in the literature [1, 5, 11, 26], features such as robustness, adaptability, scalability, and resource efficiency, are essential for Multi-UAV systems, but the focus of our work is the characteristics of modularity and reusability, ultimately targeting maintainability.

Therefore, this work presents a reusable software architecture and a framework focusing on reuse and modularity. In this way, we seek to encourage the development of Multi-UAV Cooperative Systems by reducing development costs and time, as well as promoting quality in the long term. Our architecture intends to be general for implementing Multi-UAV cooperative systems, enabling carrying out different missions. For this, we propose the Mysterio Framework to develop and assess this Multi-UAV architecture with the support of simulated environments.

The remaining sections are organized as follows. Section 2 presents relevant concepts for understanding software architecture for UAVs. Section 3 presents the related works. Section 4 presents the proposed architecture and framework. In Sections 5 and Section 6, we detail the framework instantiation process and present instances of our framework, respectively. Finally, Section 7 concludes the paper.

2 SOFTWARE ARCHITECTURE FOR UAVS

UAVs, popularly known as drones, have important characteristics, such as the ease of deployment, high flight capacity, and the ability to hover in the air [12], as well as properties such as robustness, adaptability, resource efficiency, scalability, cooperation, heterogeneity and self-configuration [34].

Unlike using a single UAV that has limited capabilities, the coordination and cooperation of multiple UAVs can create a system capable of amplifying its advantages. Such cooperation is commonly defined with the idea of imitating animal behavior [20], using the so-called swarm algorithms [3, 20].

As the size and complexity of software and cyber-physical systems increase, software architecture becomes a critical success factor. Bass *et al* [4] describe software architecture as "*the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both*". Software architecture can be also understood as an abstraction of a system with a certain level of details, showing some information and omitting others.

UAV systems can benefit from software architectures, as we can find architectures specifying specific scenarios or even defining new flight patterns or behaviors for UAVs [30]. Additionally, some architectural styles can be found in the UAV literature [6]. Paunicka et al [21] followed a layered architectural style, while Doherty et al. [10] developed a distributed and concentric architecture of components, allowing several service processes to occur simultaneously. Other works like [28] used hierarchical control architectures, in which the higher the level of the component, the more robust it is. In [8], the authors present a hierarchical architecture for autonomous UAVs systems. Through this multi-level hierarchy architecture, software based on it becomes more organized and modular. Furthermore, there are also behavior-based architectures as in [7] and [6], where systems behave based on the inputs (perceptions) of the detected situations. Understanding common architectural styles for UAVs is important for envisioning a reusable architecture for UAV systems. In addition, the selection of communication protocols is also an important architectural decision.

3 RELATED WORKS

In the literature, we can identify related works on Multi-UAV systems describing their software architectures. However, unlike our work, their goals do not address software reuse. For example, in [16], the authors developed a Multi-UAV system with a software architecture focused on communication networks that would interconnect UAVs, base stations, ground control station, satellite, and wireless sensor networks. The authors explained the Multi-UAV system has several means of communication so that the UAVs could deal with the scenario of monitoring and controlling risk areas. Unlike this proposal, our work has a general-purpose software architecture focused on modularization and reuse for the development of Multi-UAV systems, that is, it is not limited to specific application scenarios.

In [30], the authors presented a software architecture platform aimed at autonomous vision-based navigation, obstacle avoidance, and convoy tracking. The system's UAVs make their own flight decisions to avoid obstacles depending on the data collected by their flight sensors. Unlike our work, the authors described their general architecture for Multi-agent systems, so they used UAVs to carry out missions with a focus on autonomous and collaborative control. In some specific missions of other work, the authors developed a multi-agent system using UAVs. Unlike our work, this one focused on developing a multi-agent framework aimed at planning paths to optimize information-based objective functions. With this, it was possible to include individual controllers, sensors, and user interfaces to carry out missions and improve the path planning scheme [31].

Among the works found in the literature, some were fundamental for the development of our architecture [2, 9, 13, 15, 16, 18, 21, 23, 25, 30, 31, 34]. These works are presented in Table 1. All of them developed UAV systems in their work and presented and discussed their software architectures. Mostly, these architectures follow the layers style.

Finally, the architecture presented in [2] reuses a software architecture presented in other works by the same authors to develop a framework specific for Mobile Autonomous Systems with a focus on autonomy and individual or team self-management of each mobile system. In this approach, the mobile systems used were UAVs. Unlike our work, this work specialized not only in UAVs, but in mobile systems. In addition, the authors carried out a study on the behavior and interactions of the mobile systems (UAVs) developed by them. The authors also used distributed management policies and focused on optimizing the self-management of UAVs in performing distributed policy-based tasks.

4 THE MYSTERIO ARCHITECTURE

In this section, we present a software architecture and a framework (called Mysterio) focused on modularity and reuse aspects for the development of cooperative Multi-UAV Systems. In this way, our efforts in the development of the framework were to implement the Mysterio architecture following software architecture and design principles. The architectural style of our architecture can be seen as layered architecture.

Analyzing our architecture from the bottom up perspective (in Figure 1), the cyber-physical systems of the UAVs is peripheral in the bottom, the system controller in the middle, and the Framework Client at the top. The system controller (core) starts with the database at the bottom. Then, on a layer above, there is a Communication Bridge, the Repository, and subsequently the Status Manager. Further up, there are the mission and task components (Task, Task Manager, Mission Planner) until reaching in MysterioFacade and Framework Client. To assess the proposed architecture, we implemented four instances of Multi-UAV systems using the framework, communicating with a virtual environment using the OMNeT++ simulator.[1] simulator.

The Mysterio framework aims to provide a reusable software architecture for building cooperative Multi-UAV systems. This way, it should be structured independently from the internal implementation of the involved UAVs (real or virtual), providing simple interfaces to communicate with them. Thus, it does not include support for developing the UAV's internal architecture, which could be achieved with other solutions like [22], focusing on the systemic linkage and controls.

For the design of the Mysterio Architecture, an analysis of several works found in the literature was first carried out, among them we selected the related works in Table 1 because these works described and showed the design of their software architectures for UAVs. We captured useful information from the architectures and their components. Not all related works in this work presented the architectural design, so it was not possible to verify them, only the ones in the table. With all the relevant data extracted from each work, we organized the most common and non-common components and responsibilities into groups. When analyzing these groups, we identified that the non-common information was specific to each application scenario. Thus, creating components that were only useful for a specific scenario would not be consistent with the idea of reusable software architecture.

[1] https://omnetpp.org/

After the analysis, with the most common and recurring information from the architectures, we transformed them into responsibilities that every Multi-UAV system would need to have. Responsibilities such as: managing the status of UAVs, carrying out missions, creating tasks, and other responsibilities. With this set of responsibilities, we designed components representing those responsibilities and assembled our architecture composed of all the components coming from these responsibilities. Table 1 associates responsibilities with components of the Multi-UAV architectures existing in the literature. Each column refers to the architecture components of some related work and each row refers to a responsibility. As expected, we cannot observe full coverage of responsibilities for all architectures. In some works, the authors discuss these responsibilities, but others do not. The focus of these architectures was to solve specific problems of their scenarios. Thus, they do not concern about basic responsibilities of Multi-UAV systems or do not have a particular module to handle such responsibility. Given this, our general-purpose architecture covers all the responsibilities understood as essential (core) for Multi-UAV systems.

The development of Multi-UAV systems using the proposed architecture requires a base station to control the system and UAVs with support for network communication. The proposed architecture is agnostic regarding the UAVs model if they are real or virtual, heterogeneous or not. In the architecture, the base station is the control center of the Multi-UAV System and, through it, mission control, communication, status collection, and task assignment are performed for all UAVs in the system.

According to the Mysterio architecture (Figure 1), the component **Communication Bridge** is responsible for all communication between the framework and the UAVs, providing an interface with default methods to allow connecting and disconnecting UAVs to the framework, as well as sending and receiving messages. The implementation of this component is separated into three classes: i) the UAVProbe class (deployed inside the UAV as a proxy) is responsible for communicating with the framework and it must use the same communication protocol of the Multi-UAV system to communicate; ii) when the UAVs communicate with Mysterio, they send messages to the Communication class, which forwards received messages to other components, so they can handle each specific type of information; iii) in turn, these components such as the Status Manager or Task Manager, need to extend the Communicable class.

Another important component of the architecture is the Status Manager, which is responsible for managing the status information of the UAVs. In a Multi-UAV system, the general framework needs to monitor data from the UAVs and, by default, the Status Manager collects information regarding battery, flight time, altitude and location/geographical position, speed, payload, availability (communication or tasks), and idleness. In the case of heterogeneous UAVs, some UAVs may not support all listed properties, and the application may also need to extend this class for additional properties.

One way to deal with the various status information is to use a unified status class. This is the approach we use in the Mysterio instances explained in the section 6, where each information is considered as status must become an attribute. If the developer using

Table 1: Essential responsibilities and related components of existing Multi-UAV architectures.

	Components in Related Works											
Related Works / Responsibilities	[2]	[9]	[31]	[13]	[16]	[18]	[25]	[30]	[34]	[15]	[23]	[21]
Self-management (individual)	UAV Control Software	MUAV	Autoplot (dune)	UAVRemote Controller	UAV Controller	Described in text	Mission Coordination	UAV	Mission Control	UAVs	Autoplot	Vehicle Controller
Self-management (collective)	Team Layer	MUAV	Decision Support System	Control Station	Ground Control	Described in text	Mission Coordination /Planning	Described in text	Mission Control /Planning	Described in text	Team Control	Vehicle Mode Manager
Specify/manage tasks and missions	Mission	Mission Control	Operator	Mission Planner	Mission Control /Task Manager	Task Requester	Mission Coordination	Planning and Task Allocation	Mission Control /Planning	Mission Manager	Team Control	Resource Manager
Authenticate UAVs in communication	Discovery	Drone Access Service	No discuss/ mention	Security & identify	Just discuss about	APIs	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	Application Module	No discuss/ mention
Manage messages and communication (UAVs, other node and control station)	Comm. Layer	Comm., IDL, DGSL	Comm. (IMC)	Comm. System	Terrestrial/ Satellite Network Layer	Broker	Comm. Network	No discuss/ mention	Comm. Network	Comm. Layer	Team Control	Event Channel
Assign and change leader responsibility for some UAV	Optimizer	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	No discuss/ mention	Process Monitor	No discuss/ mention
Manages status and information of all UAVs	State Aggregator	Mission Control	Ground Station	UAV (abstraction)	Ground Control	Just discuss about	View Basestation	Planning / Task Allocation	Just discuss about	System State	Operator	Controller Comp.
Use of sensors	Just discuss about	Sensors	Autoplot	Sensor Manager	Sensor Unit	UAV /Resource	Sensing	Sensors	Sensing	Hardware Abstraction Layer	Hardware Abstraction Layer	Resource Manager
System data management	Mission Layer	Data Management	Just discuss about	Storage & Data Tools	Database	DBMS	Sensor Data Analysis	DataHub	Sensor Data Analysis	Shared Cooperative Memory	DataHub	Just discuss about
Path planning	No discuss/ mention	Route Planning	Path Planning agent	Flight controller	just discuss about	just discuss about	Mission Planning	Trajectory Tracking	Mission Planning	Mission Manager	No discuss /mention	No discuss / mention

Mysterio prefers, there is a status base class present in the framework. Extending this class is an alternative for dealing with Status in Mysterio, as it already implements the Status Manager interface. In this way, the developer only needs to add the new attributes or methods to the subclass. This subclass should be according to possible groupings of status attributes, avoiding implementing too many classes because of the number of status attributes. Furthermore, reuse would avoid a complete (re)implementation of the Status Manager interface.

The Task Manager is responsible for managing and receiving task information. The Tasks component represents tasks assigned to the UAVs, which can be approached/implemented in two different ways. One way is to handle built-in pre-programmed tasks, in which the behavior of UAVs is already implemented and the tasks passed from the framework to the UAVs contain just the information necessary for the UAV to interpret the built-in actions, the location, and time to execute them. The other way would be using command attributes in the task, where the commands are described in a language (like a DSL) the UAVs understand and execute as indicated. Both ways are implemented in the Tasks component and available in the framework for the programmer.

Mission Planner handles missions and tasks. This component has autonomy in mission management. A mission is made up of a list of tasks assigned to UAVs. This component provides a default implementation, not requiring the framework programmer to code additional implementation unless there is some specific behavior like algorithms tasks prioritization and orchestration.

In general terms, the developer uses our architecture to assign tasks to the UAVs following the flow of a mission: i) starting from the instantiation of a set of tasks; ii) the UAVs must be selected to be assigned in the system; then, iii) the tasks are dispatched to the UAVs through of the communication component; iv) the tasks are performed by the UAVs that later return the task with a status of finished.

The Repository is responsible for handling the base station database. This component provides an interface with the methods that must be implemented to manage relevant data for missions through the Mysterio framework. It is responsible for data persistence, both retrieving data from and storing data in the database. By default, an implementation of the repository interface persists mission information (such as identifier, involved UAVs, date and time), status data, and tasks assigned to each UAV. This information must be persisted in the selected database, therefore, it is up to the architect to choose an appropriate Database Management System. To persist specific information for each instance, the developer must add specific methods, by extending the implementation class and/or the repository interface.

Finally, the framework has the default RepositoryMySQL class that implements all the methods of the Repository interface for the MySQL database. This class is available in the Mysterio Framework class set, but the developer must create another class to handle another DBMS if desired. Finally, it is worth noting that RepositoryMySQL was widely reused in the framework instances, written in Section 6.

5 FRAMEWORK INSTANTIATION

In Mysterio Framework, the Communication Bridge and Status Manager components are interfaces that the developer of an instance of the framework must implement in their way, however, in the framework's class set, there is a status class implemented, in case the user wants to extend it.

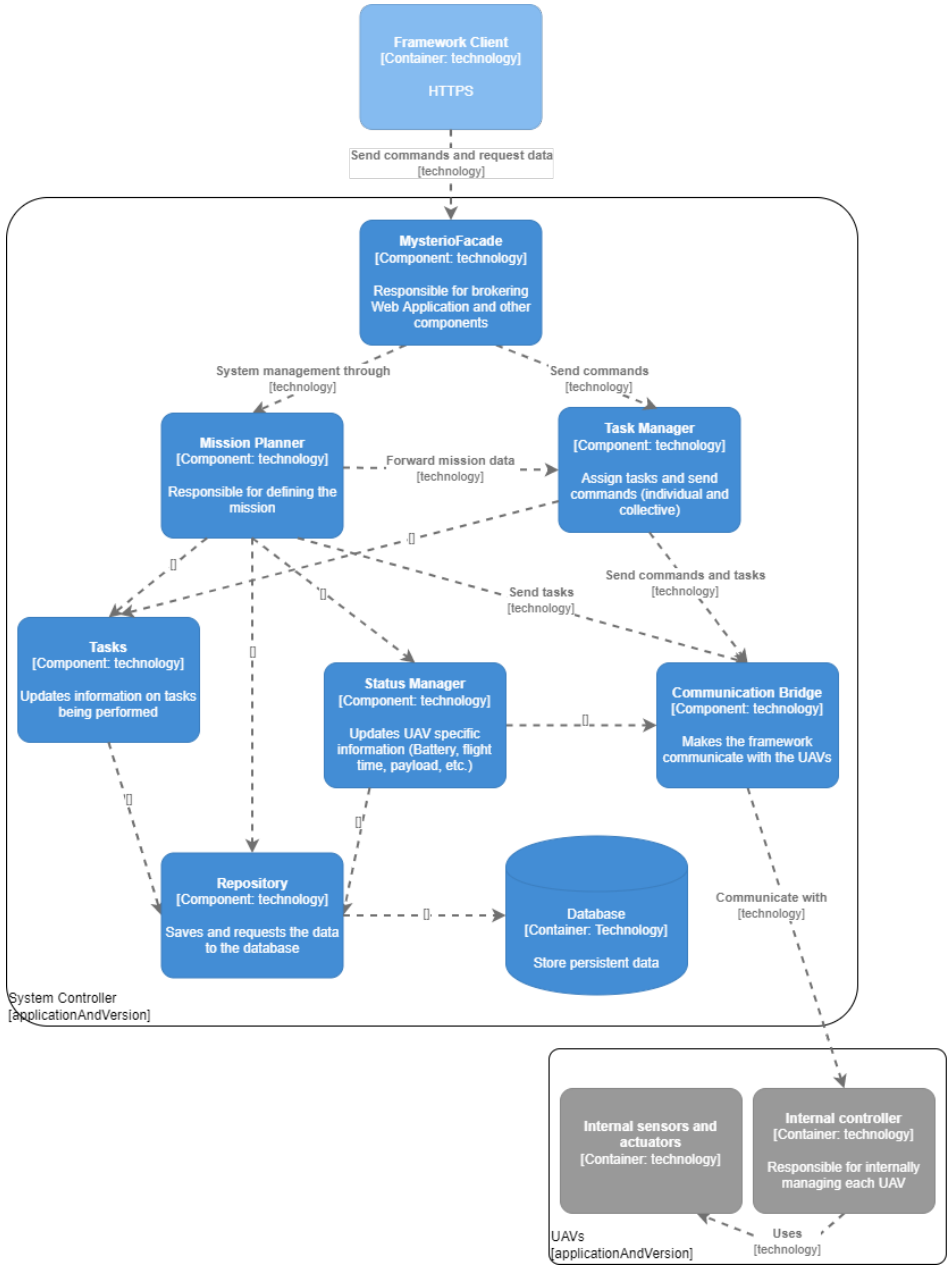

Figure 1: Component-level diagram

In applications based on the Mysterio framework, the process of handling and assigning each task is centralized so the creation and certification of a task start at the Task Manager. This component assigns an identification for each task and also verifies if a task is valid, i.e., when it has already been certified (already have a valid Task ID). But if its basic properties (task type, task ID, or assigned UAV) are changed by another component that is not Task Manager, this task must be considered invalid. This way, through such verification, it is possible to know if a given task has been compromised. Another component handling task is the Mission Planner. This component holds all valid tasks instantiated by that application.Furthermore, this component is responsible for forwarding the tasks in the framework, through the communication component, to the UAVs.

The flow of a mission in Mysterio is as follows: In this step, the instance developer passes the task type and selected UAV to the Task Manager component to create task and assign it. Starting with the Task Manager, this component creates the requested task using the provided information. Then, this component automatically creates and associates an ID for the task. After, the Task Manager sends the

task to the Mission Planner component, which manages all tasks in the mission and assigns them to the UAV. After that, Mysterio dispatches these tasks through the communication component to the corresponding UAVs and it receives status updates for each task, especially when tasks are completed.

The implementation of the framework client and MysterioFacade is at the user's discretion. This implementation should be done at the end of the instantiation process to integrate all the other components already implemented. In our architecture, MysterioFacade is the intermediary between the other components with the Framework Client, facilitating the integration with it. This way, the developer is adapting the components that integrate Mysterio with a client/an interface that better fits the needs of the user of the Multi-UAV system. The other components, such as Task Manager, Mission Planner, and Tasks have their default implementation provided by Mysterio, but it is not an obstacle for the developer to extend and customize these classes.

The Repository, for example, is another interface that must be instantiated by the developer according to the selected database, as it works as an abstraction layer between the other components and the database. Using this component, the mission, task, and status data of UAVs are required to be persisted to manage UAVs. Other types of data can be persisted depending on the application's need or scenario. In Section 6, we present the created instances. In these, RepositoryMySQL was widely reused for data persistence.

In Figure 2, the Mysterio framework instantiation process is illustrated. It starts with the implementation of the communication interfaces. Then, the extending or full reuse of the status classes, and after the developer reuses or implements mission and tasks components (Tasks, Task Manager, and Mission Planner). Finally, the developer should extend the MysterioFacade with the additional operations that will be exposed to the framework client, and implement or reuse the Repository classes. White blocks represent steps the programmer implements the provided interfaces and has mostly his code. Blue blocks are steps for extending and reusing the code already implemented in the framework.

6 EXAMPLES OF MYSTERIO INSTANCES

In addition to proposing the architecture and framework, we the same authors developed four instances to work to support understanding and to assess the framework's feasibility. All these work through simulations with UAVs performing missions and tasks in virtual scenarios. The simulator used for virtual simulations was OMNeT++ version 5.4.1 with INET version 4.0. The framework and instances are available on GitHub [2]. The operating system running the simulation was Ubuntu 18.04 LTS and the machine running the whole system is an Intel Core i7-10750H CPU 2.60GHz x 6 and 4GB of RAM.

To test each instance, a scenario referring to an open environment was built, with a car stopped inside the scenario for instances Connor and Electro, in addition to a sheep for the Marko instance. In the system, some UAVs were responsible for mapping the place, while one was responsible for covering the target (car or sheep) at certain times.

[2]https://github.com/savionasc/mysterio

Figure 3 shows the features present in each framework instance. From the first to the last instance, we can see the increasing complexity of the features mentioned. Such increasing complexity allowed us to test the development capabilities of Multi-UAV systems the framework provides. The system developed for the Connor instance is less complex than the systems developed for later instances. Connor's scenario didn't require much of the system to be developed. In this way, the system had homogeneous UAVs, tasks without requirements and replanning, as well as missions without ordering between tasks. In this case, the programmer did not need to develop such a robust system and used UAVs without distinction of capabilities, where each UAV is capable of performing any assigned task.

In Electro, Marko and Osborn instances, the developed system was able to use heterogeneous UAVs, as the programmer made it more intelligent and equipped with many capabilities. In this way, the systems of these instances made it possible to create tasks with requirements to be achieved by a UAV. In addition, these systems made it possible to perform the re-planning of tasks, to know when an UAV or a group of them could or could not perform a certain task. Not all UAVs can be enabled to perform a task due to their limitations or that of other UAVs. Such systems also allowed for more elaborate missions with tasks that follow the dependency order (one task can only be performed after another) in which they were determined. Finally, synchronous tasks are also supported by the framework. It is worth noting that synchronous tasks were used in the Multi-UAV system developed in the Osborn instance presented in section 6.4.

The instances were built iteratively, evaluating the framework, increasing the complexity and challenges of the scenarios. Thus, our evaluations sought to show the capabilities provided by the Mysterio framework in the instances. Given that, the instances solve important/relevant problems of real applications such as: swarm control (in both), patrol (Connors and Electro), search and rescue (in Marko), formation flight (Osborn), Consensus algorithms (Osborn). The completeness level of a Multi-UAV system varies according to the complexity of the problem the system needs to solve.

6.1 The Connor instance

In this instance, two similar UAVs with the same hardware and system technological capabilities, that is, UAVs without hierarchical or technological differences were selected to compose the Multi-UAV system developed through the framework. These UAVs were controlled by functions provided by the framework on the base station that was called remotely through the Framework Client (control interface) that allowed the UAVs to perform mission tasks. The scenario had only an open field with the presence of UAVs and a car parked near the center of the scenario. Each UAV started by covering the environment until the assignment of tasks to perform. For one of the UAVs, it was assigned a task to circulate the car in the scenario. The second UAV that was also covering the scene was given a similar task, but to traverse the entire environment changing the flight pattern around the entire patrol-shaped area on the block. From this instance, it was possible to build a Multi-UAV System predominantly reusing the codes already available in the framework.

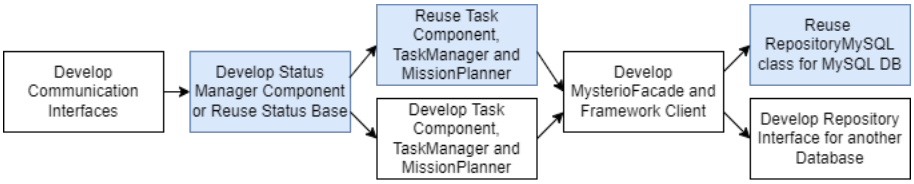

Figure 2: Framework instantiation process.

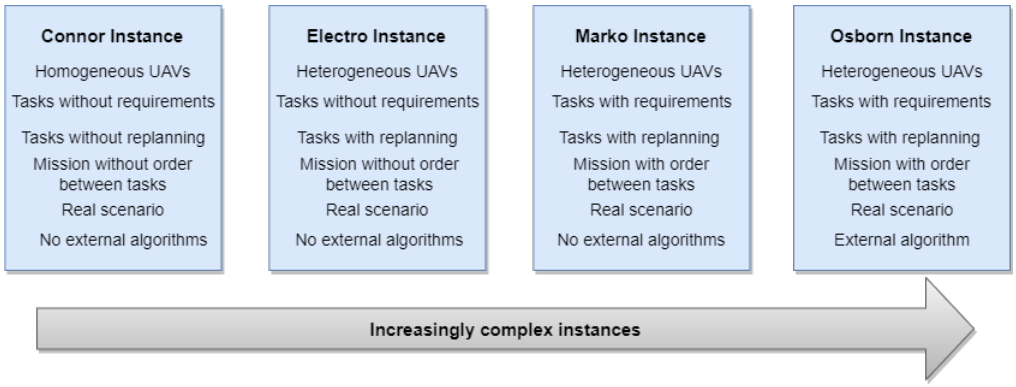

Figure 3: Main characteristics of the instances developed with Mysterio.

6.2 The Electro instance

In this instance, two UAVs were used that perform tasks assigned by the base station (framework), but the UAVs used are heterogeneous, unlike the Connor instance that used homogeneous UAVs, as shown in section 6.1. In this second instance, Electro, the first UAV received a common UAV capability and the second had its subordinate capability, where it was practically on standby. In the system, the subordinate keeps waiting for any communication; and it is always able to replace the next UAV that needs to be replaced, either when that UAV's battery is low, or forcefully when the framework orders and sends the task to the subordinate to take the first UAV's place in whichever task the first one is executing. In this scenario, the first UAV performs tasks similar to the tasks in the previous scenario, covering the environment and around the car, until its battery is close to fully discharged. Then, it alerts the system/base station about its battery status and returns to the base station. Predictably, the system orders the replacement UAV to perform the task, which immediately gets ready to execute the task. After performing the task, the UAV returns to the base station and waits for another replacement request. In this instance, the entire architecture and framework codes were reused and new codes were produced in a way that follows the same Mysterio architecture, showing its versatility in accepting new codes and solving problems in the scenario activities.

Finally, figure 4 presents a collage of five images of the UAV's performance. Images are numbered to indicate their order. In images 1 and 2, the UAV[0] is going around the car, and the UAV[1] is on standby awaiting orders from the base station, as it has no tasks to perform. As the UAV[0] was executing its task, its battery was low and it sent an alert message to the base station. In image 3, the UAV[0] makes an emergency landing in a predefined position by the base station and is replaced by the UAV[1]. The base station sent the task already started to the UAV[1], that is, this UAV resumed the task in the same position and state in which the task was, and continued until the task was completed (images 4 and 5 in Figure 4).

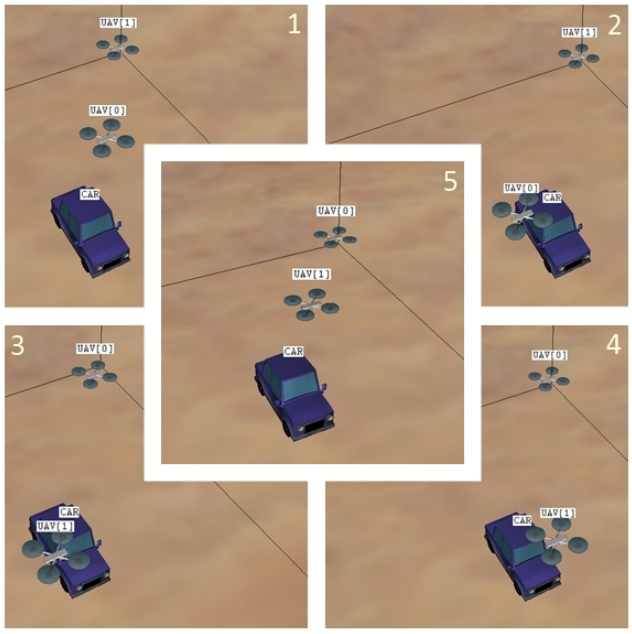

Figure 4: Electro Instance: UAV[1] replacing UAV[0].

6.3 The Marko instance

In the third instance (Figure 5), five UAVs (four workers and one parent) and a sheep were present in the scenario. The user assigns at the base station only one task for the UAV parent, to fetch the sheep and command the workers to surround it. Initially, the parent UAV starts doing its search until it finds the target (sheep). Then, it communicates with the workers and proceeds. When approaching the sheep, it forces the sheep to stop until the other UAVs surround it. In this instance, all the internal capabilities of the framework were reused, extending and generating new code for all classes from the base structure. It is worth mentioning that for this instance we have more instance-specific code, that is, user code. This way, the application is using all the components of the framework by extending abstract classes or implementing interfaces, with no direct call.

Figure 5: Marko Instance: Mission to find and surround a target.

6.4 The Osborn instance

We developed a fourth instance, called Osborn, using the Mysterio framework to implement a Cooperative Multi-UAV system. This instance implements a reference scenario as in [17]. The authors proposed a consensus-based collision-avoidance strategy that is based on an artificial potential approach and is tested in flight and formation scenarios. For that, the authors used four UAVs, being one leader and three workers, where worker UAVs flew following the leader. The UAVs flew in formation and ran the consensus algorithm and artificial potential fields to avoid collisions, providing escape solutions for UAVs flying vertically.

In our instance, we use ten UAVs, being a parent and nine workers. When running the Multi-UAV system in the scenario, the user needs to assign through the framework (representing a base station) a given group formation and flight tasks to the UAVs. At any time, the user can order the UAVs to change formation. External to the framework, UAVs use the consensus algorithm to avoid collision, as described in the related work. This consensus algorithm helped the UAVs to avoid collisions with each other, during flights and formations. In addition, the Multi-UAVs system developed was able to deal with the same scenario described in his work. Figure 6 shows two different times when UAVs entered two distinct formations.

To reach consensus, the UAVs needed to communicate with each other to prevent possible collisions. Furthermore, in the scenario, UAVs always fly respecting the assigned formation. In this instance, the Mysterio Framework is not only achieving its goals through reuse to develop new Cooperative Multi-UAVs systems, but also extending its structure to accommodate external or complementary algorithms.

Figure 6: Osborn Instance: UAVs in 2 different formations.

In the implementation of instances (Connor, Electro, Marko and Osborn), where their source code is mostly user code, just a little portion of new code was added and a good amount of lines were reused when compared to the system's class structure already implemented in the Mysterio framework. This is because Mysterio architecture provides an entire coding base for classes, as expected that for each scenario, problem, or user need, specific extensions and additional modifications must be produced in Mysterio-derived classes. In Table 2, we present data from Mysterio Framework and each instance. The table columns are showing the number of classes (# Classes), the count of lines of code (LoC). Finally, the last column shows how many lines were reused from the Mysterio Framework in each instance and the percentage of lines reused (Reused LoC (%)).

Instance	# Classes	LoC	Reused LoC (%)
Mysterio	28	1250	-
Connor	56	2293	1120 (48,84%)
Electro	60	2593	1185 (45,70%)
Marko	61	2814	1212 (43,07%)
Osborn	64	3320	1250 (37,65%)

Table 2: Reused code per instance.

Finally, each instance used all the components of the framework, evidencing the usefulness of the components of the framework architecture. Regarding the use of instances as a preliminary evaluation, we achieved positive results, because in all instances we could successfully develop a cooperative Multi-UAV System, reusing the base code. In Table 2, a considerable part of the code for each instance came from the code provided by the Mysterio Framework. As expected, most components are essential for building a cooperative Multi-UAV system, in this specific case of our instances, all components modeled by the architecture were reused. The proposed architecture does not intend to represent all the necessary components for all possible Multi-UAV systems, since each Multi-UAV System has unique scenarios, problems, or characteristics that may require additional components.

Finally, by analyzing reuse percentages in depth, we observed the components that most cover reuse percentages are the communication (Communication Bridge), task (Task Manager, Tasks) and mission (Mission Planner) components. The percentage of reuse decreases as the complexity and lines of code of the instances increase (Table 2). This way, the framework does not specialize to specific problems, but to what is essential for Multi-UAV Cooperative Systems. Furthermore, the Mysterio architecture and framework

support extensions and adaptations allowing the user to build their systems in a structured way.

7 CONCLUSIONS

Due to the importance of Multi-UAV systems, there is a need for supporting the architectural design and reuse when developing cooperative Multi-UAV systems.

Along with the motivation of our work, the software architecture and all the framework code developed in this research are open and available to be reused by the scientific community. Through the four instances developed and presented in Section 6, we understand that we have achieved a starting point of architecture and a framework enabling the development of Cooperative Multi-UAV Systems, fostering design and code reuse. We hope that our architecture and framework serve as a basis for developers to reuse and develop their cooperative Multi-UAV systems or even evolve the Mysterio framework. This way, developers do not need to develop these systems from scratch, saving time and effort.

As future work, we intend to assess its use with knowledgeable programmers. Thus, we avoid implementation biases of Mysterio programmers in the instances. It would be interesting to see how other developers would evaluate the architecture to have the same level of reuse achieved. For another future instance, we intend to use the framework to develop Multi-UAV systems in more complex scenarios including other simulators, like CoppeliaSim (former V-Rep). Such an instance could show Mysterio's independence from the UAV platforms, not relying only on OMNet++.

REFERENCES

[1] Muhammad Yeasir Arafat and Sangman Moh. 2019. Routing protocols for unmanned aerial vehicle networks: A survey. *IEEE Access* 7 (2019), 99694–99720.

[2] Eskindir Asmare, Anandha Gopalan, Morris Sloman, Naranker Dulay, and Emil Lupu. 2012. Self-management framework for mobile autonomous systems. *Journal of Network and Systems Management* 20, 2 (2012), 244–275.

[3] Argel A Bandala, Elmer P Dadios, Ryan Rhay P Vicerra, and Laurence A Gan Lim. 2014. Swarming algorithm for unmanned aerial vehicle (uav) quadrotors–swarm behavior for aggregation, foraging, formation, and tracking–. *Journal of Advanced Computational Intelligence and Intelligent Informatics* 18, 5 (2014), 745–751.

[4] Len Bass, Paul Clements, and Rick Kazman. 2012. Software Architecture in Practice (Third Edit., p. 624).

[5] Ilker Bekmezci, Ozgur Koray Sahingoz, and Şamil Temel. 2013. Flying ad-hoc networks (FANETs): A survey. *Ad Hoc Networks* 11, 3 (2013), 1254–1270.

[6] Fred Briggs. 2012. UAV software architecture. In *Infotech@ Aerospace 2012*. Researchgate, 2539.

[7] Guowei Cai, Ben M Chen, and Tong Heng Lee. 2011. *Unmanned rotorcraft systems*. Springer Science & Business Media.

[8] Hai Chen, Xin-min Wang, and Yan Li. 2009. A survey of autonomous control for UAV. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, Vol. 2. IEEE, IEEE, 267–271.

[9] Kai Daniel, Bjoern Dusza, Andreas Lewandowski, and Christian Wietfeld. 2009. AirShield: A system-of-systems MUAV remote sensing architecture for disaster response. In *2009 3rd Annual IEEE Systems Conference*. IEEE, IEEE, 196–200.

[10] Patrick Doherty, Gösta Granlund, Krzysztof Kuchcinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. 2000. The WITAS unmanned aerial vehicle project. In *ECAI*. 747–755.

[11] Lav Gupta, Raj Jain, and Gabor Vaszkun. 2015. Survey of important issues in UAV communication networks. *IEEE Communications Surveys & Tutorials* 18, 2 (2015), 1123–1152.

[12] Samira Hayat, Evşen Yanmaz, and Raheeb Muzaffar. 2016. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys & Tutorials* 18, 4 (2016), 2624–2661.

[13] Chen Hong and Dianxi Shi. 2018. A control system architecture with cloud platform for multi-uav surveillance. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 1095–1097.

[14] Christopher-Eyk Hrabia, Axel Hessler, Yuan Xu, Jan Brehmer, and Sahin Albayrak. 2018. Efffeu project: Efficient operation of unmanned aerial vehicles for industrial fire fighters. In *Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. 33–38.

[15] Taygun Kekec, Baris Can Ustundag, Mehmet Ali Guney, Alper Yildirim, and Mustafa Unel. 2013. A modular software architecture for UAVs. In *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 4037–4042.

[16] Lobna Krichen, Mohamed Fourati, and Lamia Chaari Fourati. 2018. Communication architecture for unmanned aerial vehicle system. In *International Conference on Ad-Hoc Networks and Wireless*. Springer, Springer, 213–225.

[17] Yasuhiro Kuriki and Toru Namerikawa. 2014. Consensus-based cooperative formation control with collision avoidance for a multi-UAV system. In *2014 American Control Conference*. IEee, 2077–2082.

[18] Sara Yousif Mohamed Mahmoud and Nader Mohamed. 2015. Toward a cloud platform for UAV resources and services. In *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*. IEEE, 23–30.

[19] Naser Hossein Motlagh, Tarik Taleb, and Osama Arouk. 2016. Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives. *IEEE Internet of Things Journal* 3, 6 (2016), 899–922.

[20] Iñaki Navarro and Fernando Matía. 2012. An introduction to swarm robotics. *Isrn robotics* 2013 (2012).

[21] James L Paunicka, Brian R Mendel, and David E Corman. 2005. Open control platform: A software platform supporting advances in uav control technology. *Software-Enabled Control: Information Technology for Dynamical Systems* (2005), 39–62.

[22] BF Leonardo Ramos, B Franca, L Montechi, and E Colombini. 2018. The rocs framework to support the development of autonomous robots. Relatório Técnico. *Instituto de Computaçao. Universidade Estadual de Campinas (Unicamp), Tech. Rep* (2018).

[23] Allison Ryan, Xiao Xiao, Sivakumar Rathinam, John Tisdale, Marco Zennaro, Derek Caveney, Raja Sengupta, and J Karl Hedrick. 2006. A modular software infrastructure for distributed control of collaborating UAVs. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. 6455.

[24] B Moses Sathyaraj, Lakhmi C Jain, Anthony Finn, and S Drake. 2008. Multiple UAVs path planning algorithms: a comparative study. *Fuzzy Optimization and Decision Making* 7, 3 (2008), 257.

[25] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and Bernhard Rinner. 2015. An autonomous multi-UAV system for search and rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*. 33–38.

[26] Vishal Sharma, Navuday Sharma, and Mubashir Husain Rehmani. 2019. Control over Skies: Survivability, Coverage and Mobility Laws for Hierarchical Aerial Base Stations. *arXiv preprint arXiv:1903.03725* (2019).

[27] Giuseppe Silano and Luigi Iannelli. 2021. MAT-fly: an educational platform for simulating unmanned aerial vehicles aimed to detect and track moving objects. *IEEE Access* 9 (2021), 39333–39343.

[28] Gregory Sinsley, Lyle Long, Albert Niessner, and Joseph Horn. 2008. Intelligent systems software for unmanned air vehicles. In *46th AIAA Aerospace Sciences Meeting and Exhibit*. 871.

[29] OM Tachinina, OI Lysenko, and IV Alekseeva. 2017. Path constructing method of unmanned aerial vehicle. In *2017 IEEE 4th International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*. IEEE, IEEE, 254–258.

[30] John Tisdale, Allison Ryan, Marco Zennaro, Xiao Xiao, Derek Caveney, Siva Rathinam, J Karl Hedrick, and Raja Sengupta. 2006. The software architecture of the Berkeley UAV platform. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*. IEEE, IEEE, 1420–1425.

[31] John Patrick Tisdale. 2008. *Cooperative sensing and control with unmanned aerial vehicles*. University of California, Berkeley.

[32] Ashwin Vasudevan, D Ajith Kumar, and NS Bhuvaneswari. 2016. Precision farming using unmanned aerial and ground vehicles. In *2016 IEEE Technological Innovations in ICT for Agriculture and Rural Development (TIAR)*. IEEE, IEEE, 146–150.

[33] Patrick Vincent and Izhak Rubin. 2004. A framework and analysis for cooperative search using UAV swarms. In *Proceedings of the 2004 ACM symposium on Applied computing*. 79–86.

[34] Evşen Yanmaz, Saeed Yahyanejad, Bernhard Rinner, Hermann Hellwagner, and Christian Bettstetter. 2018. Drone networks: Communications, coordination, and sensing. *Ad Hoc Networks* 68 (2018), 1–15.

[35] Qiuyue Yu, Lei Cheng, Xin Wang, Pengxiang Bao, and Quanmin Zhu. 2018. Research on Multiple Unmanned Aerial Vehicles Area Coverage for Gas Distribution Mapping. In *2018 10th International Conference on Modelling, Identification and Control (ICMIC)*. IEEE, IEEE, 1–5.