Journal of Software Engineering Research and Development, 2023, 11:11, doi: doi.org/10.5753/jserd.2023.3314
This work is licensed under a Creative Commons Attribution 4.0 International License..

Simulation-supported development for cooperative Multi-UAV Systems with the Mysterio framework

Antônio Sávio Nascimento Cavalcante [**Universidade Estadual de Campinas**| ***savionasc@gmail.com***]

Breno Bernard Nicolau de França [**Universidade Estadual de Campinas** | ***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 innovative 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 different scenarios. Finally, we discuss the framework's potential to provide and support design and code reuse to develop Cooperative Multi-UAVs systems for different application scenarios. The results showed the potential to develop multi-UAV systems using the proposed framework. Additionally, we extend our previous work bringing conceptual evolution and advances in the architecture and the framework. Finally, this evolution extends the framework API to support computer simulations of UAV systems based on the OMNeT++ simulator. This API is suitable for Single-UAV and Multi-UAV systems and has already been adapted to communicate with base stations implemented through the Mysterio Framework.

Keywords: *MultiUAV Systems, Frameworks, Computer Simulation*

1 Introduction

Research on Unmanned Aerial Vehicles (UAVs) has emerged over the years and is becoming popular in civil and military applications (Bekmezci et al., 2013; Gupta et al., 2015). There are several examples of UAV applications for carrying out operations in different fields such as agriculture (Vasudevan et al., 2016), risky operations such as fire management (Hrabia et al., 2018), search and rescue (Scherer et al., 2015) and others. Furthermore, the use of UAVs presents several technical challenges like planning algorithms (Tachinina et al., 2017; Sathyaraj et al., 2008), area coverage and mapping (Yu et al., 2018), integration with Internet of Things solutions (Motlagh et al., 2016), 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 survivability (Gupta et al., 2015; Bekmezci et al., 2013). Another advantage is the reduction of overall execution time, given the number of nodes executing tasks (Sharma et al., 2019).

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 always available. The design of Multi-UAV systems foresees some topologies that impact architectural decisions in a Multi-UAV system (Bekmezci et al., 2013). 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's 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 (Vincent and Rubin, 2004), 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 (Briggs, 2012; Sinsley et al., 2008). The selection of technologies for the development of such an architecture is essential; that is, technologies such as hardware, frameworks, and protocols impact several system attributes.

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 (Silano and Iannelli, 2021).

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 (Arafat and Moh, 2019; Gupta et al., 2015). 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 cooperative Multi-UAV systems, the complexity of such systems increases, as well as the need to organize and modularize their software components, targeting functional correctness and flexible evolution. Still, the lack of technologies focusing on de-

sign and code reuse to develop this type of system makes it even more challenging, in addition to organizing them architecturally to accomplish their tasks efficiently, whether individual or collective. Thus, a poorly designed architecture in Multi-UAV systems makes them inefficient and does not exploit their expected benefits, such as the short operation time in carrying out a mission.

In the literature (Arafat and Moh, 2019; Gupta et al., 2015; Bekmezci et al., 2013; Sharma et al., 2019), features such as robustness, adaptability, scalability, and resource efficiency, are claimed as essential for Multi-UAV systems. However, maintainability is also a relevant characteristic to achieve dependability (Avizienis et al., 2004). So, our focus is on 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 and promoting quality in the long term. Our architecture intends to be general for implementing Multi-UAV cooperative systems, enabling the carrying out of different missions. For this, we propose the Mysterio Framework to develop and assess this Multi-UAV architecture with the support of simulated environments.

This paper is an extended version of (Cavalcante and De França, 2022). The main additions in this paper are as follows:

- We evolved the architecture and the Mysterio framework, including two new levels of the architecture description using the C4 model. This way, we describe our architecture at the context, container, and component levels. These additions are presented in Sections 5 and 6.
- We also include a more complete description of our research method in Section 4.
- We extended the framework API (Application Programming Interface) by providing support for computer simulations of UAV systems (Section 6), evolving the framework structure and API.
- To evaluate this extension, we implemented a new instance based on the Connor instance to assess the new integrated support for computer simulations as in Section 7.5.
- Added guidance on developing simulation-supported instances using the OMNeT++ simulator[1] in Section 7.5.

The remaining sections are organized as follows. Section 2 presents relevant concepts for understanding UAV software architectures. Section 3 presents related works. Section 4 presents the research method. Section 5 presents the proposed architecture and framework. Sections 6 and 7 detail the framework instantiation process and present a set of instances, respectively. Finally, Section 8 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 (Hayat et al., 2016), as well as properties such as robustness, adaptability, resource efficiency, scalability, cooperation, heterogeneity and self-configuration (Yanmaz et al., 2018).

Unlike using a single UAV with 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 (Navarro and Matía, 2012), using the so-called swarm algorithms (Navarro and Matía, 2012; Bandala et al., 2014).

As the size and complexity of software and cyber-physical systems increase, software architecture becomes a critical success factor. Bass et al. (2012) 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 a system abstraction with a certain level of detail, 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 (Tisdale et al., 2006). Additionally, some architectural styles can be found in the UAV literature (Briggs, 2012). Paunicka et al. (2005) followed a layered architectural style, while Doherty et al. (2000) developed a distributed and concentric architecture of components, allowing several service processes to occur simultaneously. Other works like Sinsley et al. (2008) used hierarchical control architectures, in which the higher the level of the component, the more robust it is. In (Chen et al., 2009), the authors present a hierarchical architecture for autonomous UAV 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 (Cai et al., 2011) and (Briggs, 2012), 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, selecting communication protocols is also an important architectural decision.

3 Related Works

We identified related works on Multi-UAV systems describing their software architectures. However, unlike our work, their goals do not address software reuse.

In (Krichen et al., 2018), the authors developed a Multi-UAV system with a software architecture focused on communication networks that would interconnect UAVs, base stations, ground control stations, 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 (Tisdale et al., 2006), the authors presented a software

[1] https://omnetpp.org/

Table 1. Responsibilities and related components of existing Multi-UAV architectures.

Responsibility / Related Work	Components in Related Works											
	Asmare et al. (2012)	Daniel et al. (2009)	Tisdale (2008)	Hong and Shi (2018)	Krichen et al. (2018)	Mahmoud and Mohamed (2015)	Scherer et al. (2015)	Tisdale et al. (2006)	Yanmaz et al. (2018)	Kekec et al. (2013)	Ryan et al. (2006)	Paunicka et al. (2005)
Self-management (individual)	UAV Control Software	MUAV	Autoplot (dune)	UAVRemote Controller	UAV Controller	X	Mission Coordination	UAV	Mission Control	UAVs	Autoplot	Vehicle Controller
Self-management (collective)	Team Layer	MUAV	Decision Support System	Control Station	Ground Control	X	Mission Coordination /Planning	X	Mission Control /Planning	X	Team Control	Vehicle Mode Manager
Mission specification and management	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	X	Security & identify	X	APIs	X	X	X	X	Application Module	X
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	X	Comm. Network	Comm. Layer	Team Control	Event Channel
Assign and change leader responsibility for some UAV	Optimizer	X	X	X	X	X	X	X	X	X	Process Monitor	X
Manage status and information of UAVs	State Aggregator	Mission Control	Ground Station	UAV (abstraction)	Ground Control	X	View Basestation	Planning / Task Allocation	X	System State	Operator	Controller Comp.
Use of sensors	X	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	X	Storage & Data Tools	Database	DBMS	Sensor Data Analysis	DataHub	Sensor Data Analysis	Shared Cooperative Memory	DataHub	X
Path planning	X	Route Planning	Path Planning agent	Flight controller	X	X	Mission Planning	Trajectory Tracking	Mission Planning	Mission Manager	X	X

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 (Tisdale, 2008).

Among the works found in the literature, some were fundamental for the development of our architecture (Asmare et al., 2012; Daniel et al., 2009; Tisdale, 2008; Hong and Shi, 2018; Krichen et al., 2018; Mahmoud and Mohamed, 2015; Scherer et al., 2015; Tisdale et al., 2006; Yanmaz et al., 2018; Kekec et al., 2013; Ryan et al., 2006; Paunicka et al., 2005). These works are presented in Table 1. They all 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 (Asmare et al., 2012) 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 focuses on UAVs and 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 Research Method

As the main goal regards the software architecture (Cavalcante and De França, 2022), implementing a framework based on it represents a first feasibility assessment. The reasoning is that once we show how the architecture can be used to build concrete instances, we can provide evidence on how implementing its abstract components can foster reuse in terms of structure and behavior. Furthermore, implementing concrete Multi-UAV software enables functional and non-functional evaluation of the architecture using the framework as a surrogate. Relevant quality attributes include maintainability, performance, and reliability.

Figure 1 presents the research method for achieving our goals. The process contains activities described in the following:

1. **Literature review on software architectures for UAVs:** We looked for software architecture described in the UAVs literature, as they represent the state-of-the-art and likely discuss challenges for architecting Multi-UAVs Systems. For this, we considered: the level of detail, strategies adopted to satisfy non-functional properties, use of verification and validation techniques, empirical evaluation, and use in real contexts, as all these issues affect the architectural design for Multi-UAV systems.
2. **Design of a reusable software architecture for cooperative Multi-UAV systems:** We designed a general architecture to work as a reference for the Mysterio framework and the Multi-UAV Systems developed based on it. For this step, we analyzed the outcomes of the literature review. We studied these architectures and extracted information on components and responsibilities from these architectures. We turned this information into responsibilities that our architecture would need to achieve. We created components based on these responsibilities and designed the first version of our architecture. The architecture design was based on the C4 model[2], using the three first levels: context, container, and components. Figure 2 presents the context-level diagram, which allowed us to identify the boundaries of Multi-UAV systems and their dependencies. It is essential to understand and set up communication and identify where each major system's responsibility should be placed, in addition to showing external dependencies and possible external actors. The container level presents deployment details, and lastly, at the component level, the architecture is presented in detail. As we implemented the framework, we refined and improved the architectural design. At this stage, we identified concerns about quality attributes that are relevant to these systems, such as performance and reliability, in addition to concerns about the UAVs themselves and the efficient use of resources such as energy, sensors, and actuators.
3. **Development of the Mysterio framework to support the instantiation of the designed architecture:** At this stage, we designed and developed our Framework based on our proposed architecture of Multi-UAV systems. This way, we provide a reusable asset to the scientific community, which is the Mysterio framework to support the development of cooperative Multi-UAVs systems in multiple domains.
4. **Evaluation of the framework effectiveness through implementing instances with cooperative missions shared among UAVs using a simulation environment:** This activity evaluated the proposed architecture through framework instances. Thus, we performed computer simulations in virtual scenarios using the OMNeT++ simulator to check if the Multi-UAV system could act cooperatively for the execution of missions assigned to it. Initially, it was intended to analyze whether the general objective of a system and the specific objectives of each UAV were achieved over the simulation. This assessment aimed to verify if the system was capable of overcoming the challenges of the missions to achieve its objectives. It was also important to check whether there were implementation or architectural flaws.

As an extension, this paper also adds three more activities:

1. **Evolution of the architecture and framework based**

[2] https://c4model.com

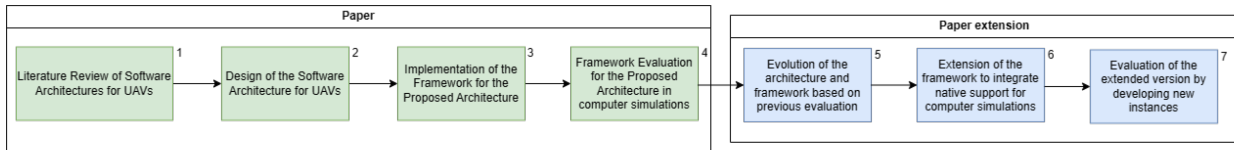

Figure 1. Research Method.

on previous evaluation: We provide an evolution of architecture and Mysterio Framework. The stage of evaluating the framework by instances was fundamental for this evolution. We have expanded our architecture by incorporating two additional levels of the C4 diagram in context and container, as we identified that the component-level architecture of the C4 diagram did not clearly indicate the appropriate placement of the framework in the architecture. We modified some framework interfaces to follow the evolved architecture.

2. **Extension of the framework to integrate native support for computer simulations**: When developing the instances presented in Section 7, we relied on computer simulations. Given the importance of computer simulation for designing and developing Multi-UAV systems, in this extended paper, we extended the Mysterio framework to foster the development of Multi-UAV systems in simulated environments, more specifically using the OMNeT++ virtual simulator.
3. **Evaluation of the extended version by developing new instances**: We developed a new instance presented in Section 7.5 to evaluate this simulation support. The result is encouraging, as we managed to reach the implementation by reusing flexible components for OMNeT++ simulations.

5 The Mysterio Architecture

In this Section, we present a software architecture and a framework called Mysterio focused on modularity and reusability for the development of cooperative Multi-UAV Systems. Our efforts in developing the framework were to implement the Mysterio architecture following solid architectural design principles. Such architecture is designed under the Layers style.

We designed components representing the responsibilities identified in Table 1 and assembled our architecture. We described the architecture using the C4 model, in which we developed context, container, and component-level models of the architecture. C4 diagrams are semantically simpler than UML models, so it may be easier for software engineers with little or no experience in UML to understand the architecture, which may be the case for UAV developers. Also, the UML diagrams corresponding to the most abstract levels of C4 (deployment and components) are less used in industry. Finally, the C4 model has gained attention in the industry.

C4 diagrams include user, containers, and internal and external components. Our context diagram was designed with a system user and more components (in blue) referring to the Multi-UAV system (framework instance) and gray components referring to the specific software of each UAV (developed outside the framework).

In the context-level diagram (Figure 2), the proposed architecture interacts with the main (human and computational) actors interacting with the Multi-UAV system. In the container-level diagram (Figure 3), we present the first refinement, showing the fundamental containers (main modules with their own execution environment) that compose the software architecture, in addition to the technology decisions. The component diagram (Figure 4) expands the containers to describe the components inside each container.

Analyzing our architecture from the bottom-up perspective (in Figure 4), the cyber-physical systems of the UAVs are peripheral at 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.

The proposal provides 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 communication interfaces. Thus, it does not include support for developing the UAV's internal architecture, which could be achieved with other solutions like Ramos et al. (2018), 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 architecture design, so it was not possible to verify them. 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.

After the analysis, with the most common and recurring information from the architectures, we distilled the responsibilities that every Multi-UAV system would need, including: managing the status of UAVs, carrying out missions, creating tasks, and others. With this set of responsibilities, we designed components representing those responsibilities and assembled our architecture composed of all the components

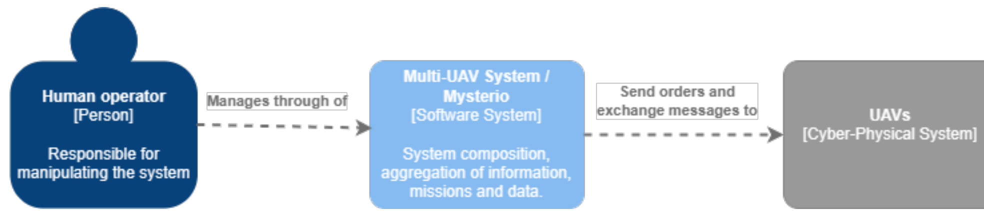

Figure 2. Context-level diagram

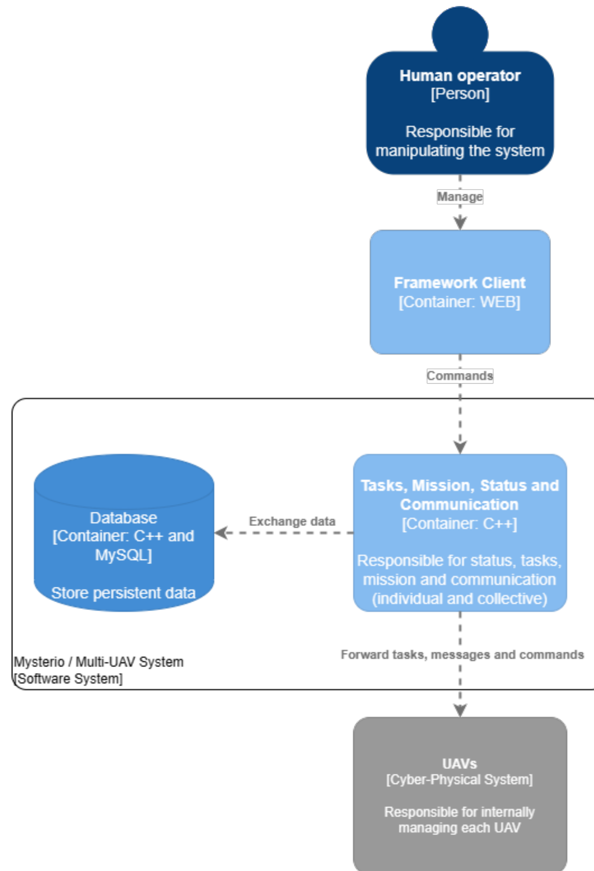

Figure 3. Container-level diagram

coming from these responsibilities.

The architecture resulting from our work has a solid set of components as it unites the strengths of existing architectures along. The proposed architecture evolved as we instantiated it for different scenarios using a framework implementing the core responsibilities. The feedback from the particular instances allowed us to improve and refine the architecture. Regarding the responsibilities, they can be found in first column of the Table 1.

Returning to Table 1, we extracted component names as in their original works. In this way, when assigning responsibilities to the related work, we unite three cases in one that does not present components. If the work does not cover a responsibility (no discussion/mention), or we cannot identify which component covers the responsibility, or even just discuss or describe it in text, but does not say which component covers the responsibility. We classify these cases in the table as ”-”.

We can mention three responsibilities that were less identified in the works analyzed to create Table 1. The first less identified was ”Assign and change leader responsibility for some UAV”. Given this, the workers opted to develop Multi-UAV systems where the UAVs worked without having a leader among them. In this way, topologies such as Star and Mesh can be applied in the configuration of the UAV System. This responsibility was rarely seen in architecture, but it is a very recurrent responsibility in the literature in general, which is why we kept this responsibility on the list. Other works mention this responsibility, but such works were not selected to compose the table, as they do not present their software architectures.

The second less identified responsibility was: ”Authenticate UAVs in communication”, we can see that this responsibility deals directly with the security of the Multi-UAV system, however, many works did not have this concern. The third least identified was ”Path planning”, this responsibility can be abstracted, depending on the application scenario and the concerns and decisions that the architecture designer takes. We can see that in search and rescue scenarios for people, animals, and objects, as well as the use of UAVs that map the environment, in some cases path planning is not required, unlike scenarios such as the delivery of packages, among others.

Other responsibilities such as ”Mission specification and management”, ”Manage messages and communica-

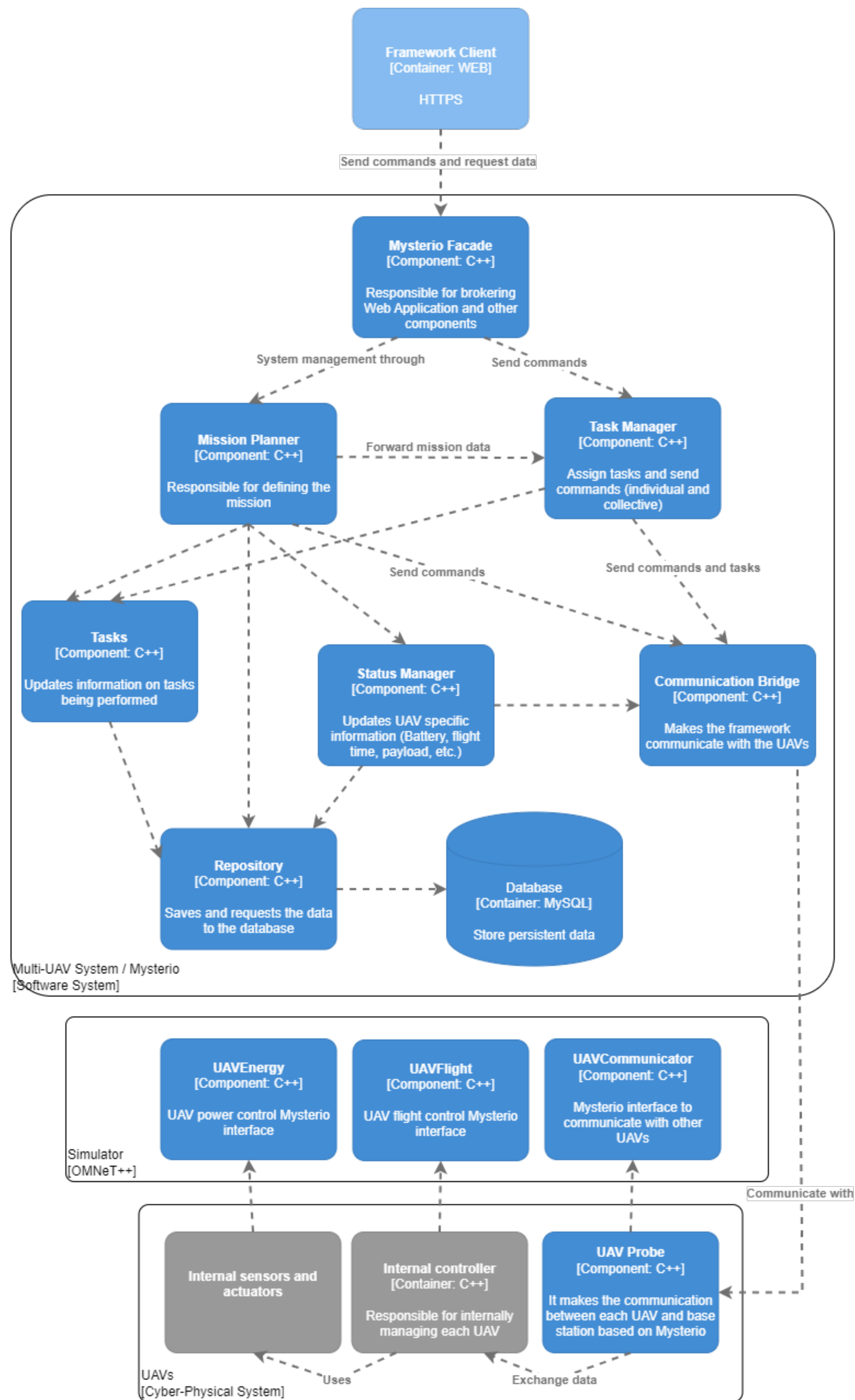

Figure 4. Component-level architecture evolved by this work.

tion (UAVs, other node and control station)" and "Self-management" were the most identified responsibilities in the cited works. These responsibilities can be considered essential, because somehow, almost all works were concerned to put in their architectures.

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.

We developed the framework in C++ since it is closer to UAV implementations. The implementation uses the object-oriented (OO) programming paradigm, with interfaces and abstract classes as the hotspots or extension points of the Mysterio framework. The framework development has been carried out in parallel with the evaluation of the framework and the technical adjustments in the architecture design. The framework evaluation used Multi-UAV systems instances to test it in specific scenarios.

According to the Mysterio architecture (Figure 4), 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 manages the UAVs' status information. 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 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 7, where each piece of information considered as status must become an attribute. If the developer using 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 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 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 the 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, implementing 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 7.

6 Framework Instantiation

This section presents the updated version of the Mysterio instantiation process based on the new extension for simulation support. Also, it presents the new API for this extension.

6.1 Instantiation Process

In the 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 5 presents a sketch of our software architecture used for instance 2 (Electro), presented in Section 7.2. Components in blue represent elements of the Mysterio framework and components in white were created specifically for the instance. In Figure 5, the MessageSender, MessageReceiver, UAVRegistry, SocketMessageReceive and SocketMessageSender are auxiliary components to the communication component. These components were created separately in the instance and they used sockets and threads at the developer's choice to facilitate separation into classes.

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 starts 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, it is possible to know if a given task has been compromised. Another component handling task is the Mission Planner which holds all valid tasks instantiated in the application. Furthermore, this component is responsible for forwarding the tasks 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 a 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 7, we present the created instances. In these, RepositoryMySQL was widely reused for data persistence.

In Figure 6, 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.

6.2 Simulation-supported Development of Multi-UAV Systems

The development of robots, including their software, demands support from simulation environments during the design time. Multi-UAV systems are not an exception as the devices may be destroyed during the tests if something does not work properly, incurring additional costs.

During the development of this work, we intensively used simulation environments to support the assessment of the architecture and the framework. It allowed us to reflect on the process and to understand that integrating simulation capabilities in the Mysterio framework is beneficial for professionals and researchers designing Multi-UAV systems. In this sense, we extended the framework with the essential constructs for defining and connecting OMNeT++ simulations.

We extended the framework to support computational simulations in OMNeT++. If the user of the framework wants to develop Multi-UAV systems in another simulator or non-virtual UAVs, these interfaces are optional. This way, we developed a new set of interfaces aimed at UAVs from computational simulations for this simulator. We provide the interface set consisting of three independent interfaces: UAVFlight, UAVCommunicator, and UAVEnergy. In Figure 5, we can see a sketch of our architecture using these new interfaces. These interfaces can be understood in Omnet++ as modules and together they represent a UAV in the computational simulation. We chose to provide these interfaces separately, as this made them independent of each other and it is up to the framework user to reuse or implement their code. The UAVCommunicator interface is responsible for managing UAV communication. It realizes the communication with the UAVs of the System, as it is linked with the UAVProbe interface making the communication between the UAV and the base station possible. The UAVFlight interface is responsible for managing the UAV's control and carrying out all of its flight mechanics. The UAVEnergy interface is responsible for managing the UAV battery level. Through OMNeT++, it is possible to carry out communication between these interfaces (as modules), in case the user of the framework wants to carry out the complete integration of the three interfaces. It is worth

Figure 5. Electro Instance Architectural Design.

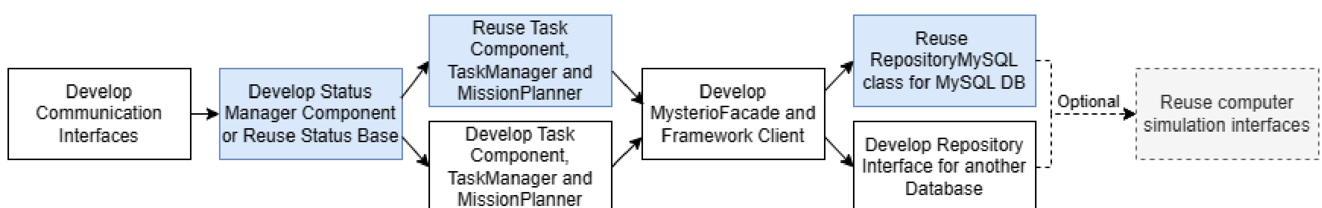

Figure 6. Framework instantiation process.

mentioning that the independency of the interfaces can eliminate the mandatory implementation of modules only in the OMNeT++ simulator. We warn that for real implementations or in another simulator, concerns such as flight control, communication, and energy management of UAVs are necessary.

In Mysterio architecture, there are two major distinct architectural configurations: with and without simulation support. An attempt to combine them into a single design may lead to misunderstanding. Therefore, it is recommended to create a single design representing the complete architecture cohesively and consistently, where we include the details of the new virtual simulation interfaces with OMNeT++. In case one is not using the OMNeT++ virtual simulator, it is not necessary to implement the simulation extensions, so disregard the simulator/OMNeT++ details in the component-level software architecture design 4. This architecture description is meant to highlight the flexibility and adaptability of these two approaches, as well as provide a more accurate understanding of the architecture and its components.

6.3 Conditions to adopt Mysterio

Having a framework and not developing a system from scratch always seems a positive alternative. Although there are some frameworks for boosting the development of single UAVs, Multi-UAV systems do not have the same level of support yet. That is the main motivation for developing the Mysterio architecture and framework. However, as with any architectural and reuse solution, adopting Mysterio as a framework also comes with conditions and trade-offs.

Regarding Mysterio's architecture, it is necessary to have a topology in which it is mandatory to have at least one base station to control the Multi-UAV System since the Mysterio components will be present mostly in the base station. Mysterio's software architecture does not cover UAV systems that do not have a base station, like in some drone swarms architectures. Furthermore, our architecture, as well as the framework, supports heterogeneous UAV nodes; however, it does not provide support for other types of vehicular or static unforeseen nodes. Extending the architecture in this sense demands additional design and development decisions.

The Mysterio framework has its interfaces implemented using the C++ programming language, as this is the technology Mysterio was implemented with. This dependency on C++ implies that developers must be familiar with the language. Also, the implementation of constituent UAVs must use or bind with C++ to deploy the UAV probes. The choice of C++ as the framework's technological base brings benefits, as it is a widely used language in technology and is commonly used in robotics and UAV systems. Finally, in terms of simulation, this support is dependent on the OMNeT++ simulator and the NED language.

7 Examples of Mysterio Instances

We developed four instances to work to support understanding and assess the framework's feasibility. All these work through simulations with UAVs performing missions and tasks in virtual scenarios. The simulator used for computer simulations was OMNeT++ version 5.4.1 with INET version 4.0. The framework and instances are available on GitHub [3]. 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 the instances Connor and Electro, in addition to a sheep for the Marko instance. In these instances, some UAVs were responsible for mapping the place, while one was responsible for covering the target (car or sheep).

Figure 7 shows the features present in each framework instance. We increase the complexity of the features in subsequent instances, allowing us to test the development capabilities for the Multi-UAV systems the framework provides. The system developed for the Connor instance is less complex than those developed for later instances. This way, the system had homogeneous UAVs, tasks without requirements and replanning, and missions with non-ordered tasks. The developer did not need to develop such a robust system and used UAVs without distinction of capabilities since each UAV can perform 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. Hence, these instances create tasks with certain requirements to be achieved by a UAV. In addition, these systems perform the re-planning of tasks and know when a 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 own or other UAVs' limitations. Such systems also allowed for more elaborate missions with tasks that follow a dependency order (one task can only be performed after another) in which they were determined. Finally, the framework also supports synchronous tasks, as used in the Osborn instance presented in Section 7.4.

The instances were built iteratively, evaluating the framework, and 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), and Consensus algorithms (Osborn). The completeness level of a Multi-UAV system varies with the complexity of the problem the system needs to solve.

7.1 The Connor instance

In this instance, two similar UAVs with the same hardware and system technological capabilities (Figure 8), 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

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

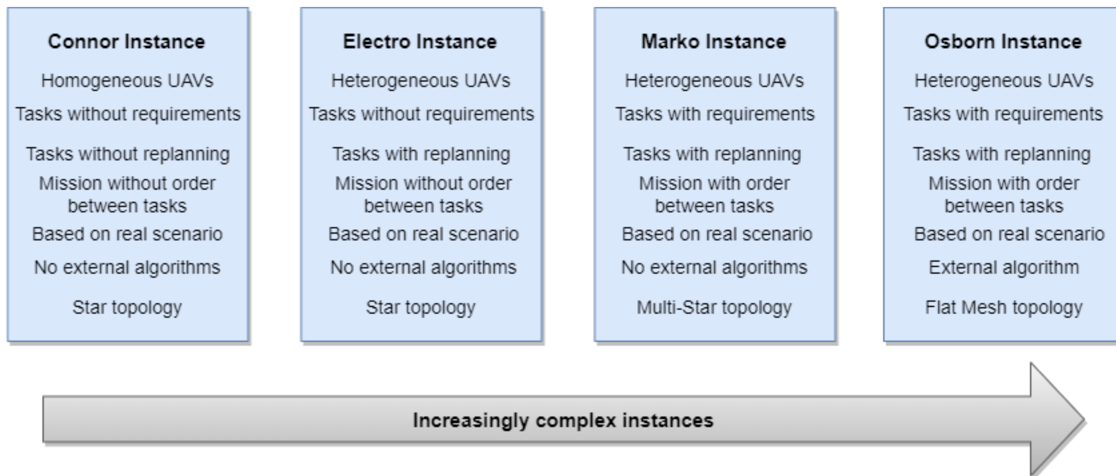

Figure 7. Main characteristics of the instances developed with Mysterio.

UAVs and a car parked near the center of the scenario. Each UAV started by covering the environment until the tasks were assigned. 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, building a Multi-UAV System predominantly reusing the codes already available in the framework was possible.

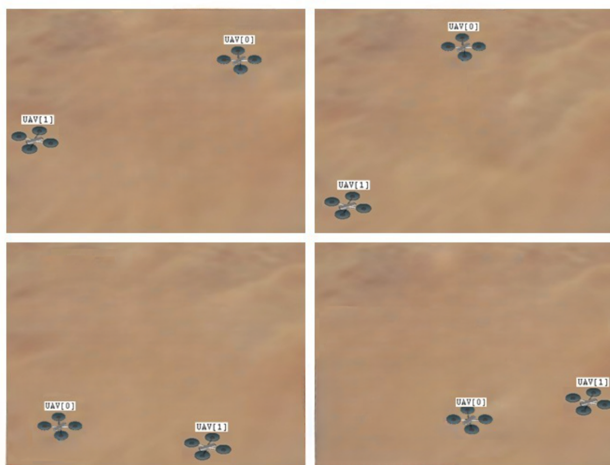

Figure 8. Connor Instance: Two UAVs performing tasks in the simulation.

7.2 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. 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

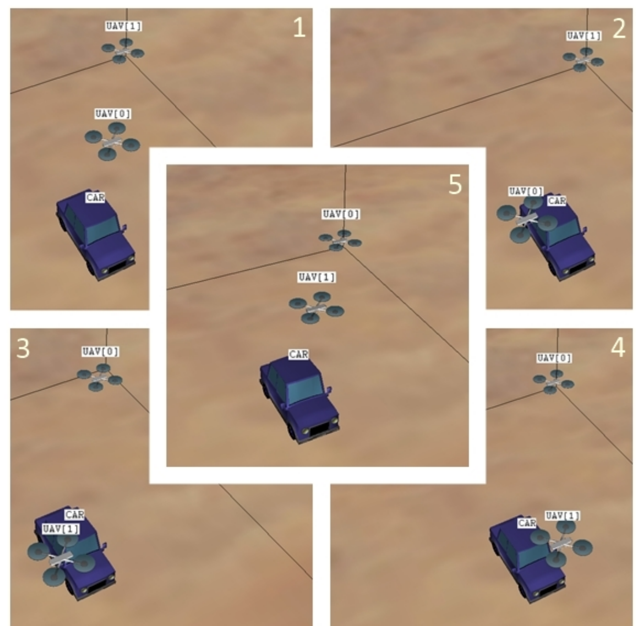

Figure 9. Electro Instance: UAV[1] replacing UAV[0].

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 9 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 9).

7.3 The Marko instance

In the third instance (Figure 10), five UAVs (four workers and one parent) and a sheep were present in the scenario. The user assigns only one task for the UAV parent at the base station, 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 uses all the framework components by extending abstract classes or implementing interfaces, with no direct call.

Figure 10. Marko Instance: Mission to find and surround a target.

When carrying out the mission of this instance, messages were exchanged between the base station and the parent UAV, as well as between the parent UAV and the worker UAVs. The OMNeT++ simulator allows recording message exchanges, so we activate this property to record message exchanges between UAVs during the execution of the simulation. Figure 11 shows the moment when the parent UAV exchanges messages between the worker UAVs to pass an order to be executed by each worker UAV. This chart was generated by a tool internal to OMNeT++ called Sequence Chart, which graphically displays the reading of data recorded on message exchanges. Through this chart, we identified that the communication of the Multi-UAV system during the computer simulation happened as planned. Other charts can be generated such as scatter charts and histograms.

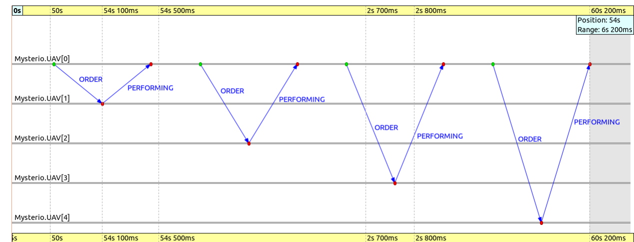

Figure 11. Message exchanges between UAVs - Marko Instance

7.4 The Osborn instance

We developed a fourth instance, called Osborn, to implement a cooperative Multi-UAV system implementing the reference scenario from (Kuriki and Namerikawa, 2014). The authors proposed a consensus-based collision-avoidance strategy based on an artificial potential approach and tested it in flight and formation scenarios. For that, the authors used four UAVs, 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.

We use ten UAVs, one parent, and nine workers in our instance. 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 collisions, 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 dealt with the same scenario described in his work. Figure 12 shows two different times when UAVs entered two distinct formations.

Figure 12. Osborn Instance: UAVs in 2 different formations.

To reach a 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 achieving its goals through reuse to develop new Cooperative Multi-UAVs systems and extending its structure to accommodate external or complementary algorithms.

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. Table 2 presents data from Mysterio Framework and each instance. The columns show the number of classes (# Classes), and 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.

Each instance used all the framework components, evidencing the usefulness of the components of the framework architecture. Regarding using 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 meaningful amount 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 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 on specific problems but on 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.

As in the UAV literature, as well as in the works of (Arafat and Moh, 2019; Gupta et al., 2015; Bekmezci et al., 2013), the choice of topology for a Multi-UAV System is fundamental. This way, we illustrate in Figure 13 which topologies were used in Mysterio instances. The Connor and Electro instances were developed following the star topology. In Marko, the system used Multi-Star, and Osborn adopts the Flat Mesh topology. In addition, it is worth noting that we were able to develop Multi-UAV systems in three classic topologies. It is important to mention that the Mysterio Framework is not topology-specific, so other topologies can be developed through the framework.

7.5 Simulation-supported Instance

For this, we implement one new instance based on the Connor instance from Section 7. In this case, our purpose was to promote an instance dedicated to virtual UAVs, so we used our new extension focused on simulation in OMNeT++.

In this instance, we use an OMNeT++ property useful for simulating Heterogeneous Multi-UAV Systems. Through this property, we can choose whether the simulated UAVs use unified files representing the UAVs or separate files. Using unified files brings some advantages such as:

- Control of UAVs in a grouped way;
- Minor code to be developed;
- More agility in the development of simulations.

We also identified a disadvantage when using unified files, it makes the code conversion to real UAVs more complex if the user of the framework wants it. Using the property with separate files brings the following identified advantages:

- The UAV code is individual and easy to understand;
- Require less complexity in adapting the code for real UAVs.

In addition, we identified some disadvantages of using the property in this way, as it requires more code for the programmer to develop and brings less agility in the development of computer simulations.

It is important to understand that this property is internal to the OMNeT++ simulator, but it has not been exposed in the literature and the examples found on the internet. It allows the programmer to decide the best way to implement computational simulations in two different ways. The first way allows standardizing the UAV simulation files, but the code implemented by the programmer must know how to deal with different UAVs, situations, and behaviors when dealing with heterogeneous UAVs. The second way is to define different files for different UAVs. This way, the user specifies what each UAV specifically can do. Both forms are useful, but each has advantages and disadvantages to analyze in simulation modeling.

Through this work, we also produced a specification entitled ”Guidelines for simulating Multi-UAVs with 'OMNeT++.” This guide is available on GitHub, and to access it and see more information, access the link available at: `https://github.com/savionasc/mysterio/tree/main/guide/`. Through this guide, we address some simulation issues, but it is worth noting that the focus is more specifically on OMNeT++. From the topics in the guide, topics related to simulations in general, features provided for simulation of multi-UAV systems using OMNeT++, how to define topologies (Figure 7 of Section 7), and choosing between discrete and continuous simulations. We also provide instructions for the main functionalities that can and are useful for Multi-UAV Systems simulations in this simulator. We also provide details on exporting simulation data and choosing the topology for the Multi-UAV System among the classic topologies (star, multi-star, flat mesh e hierarchical mesh)

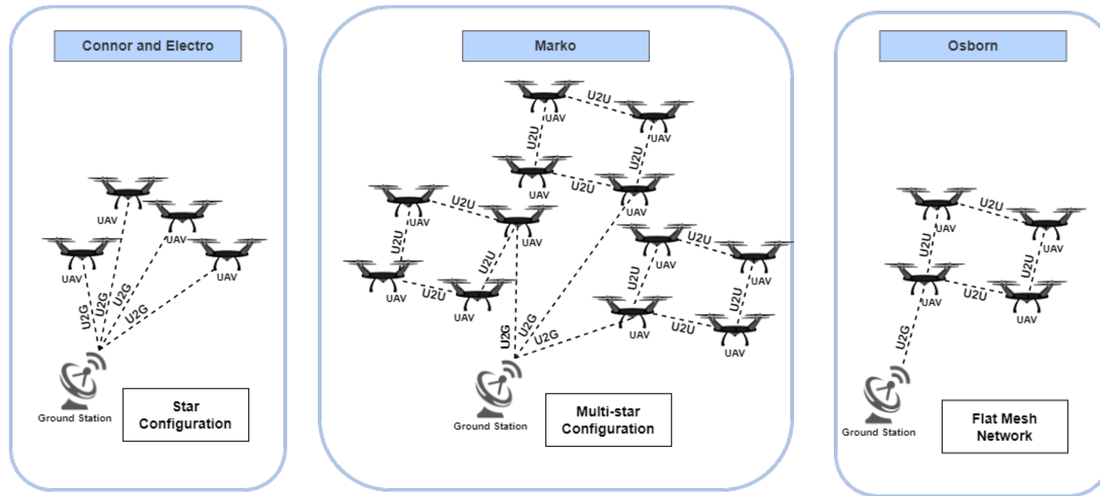

Figure 13. Instances topologies

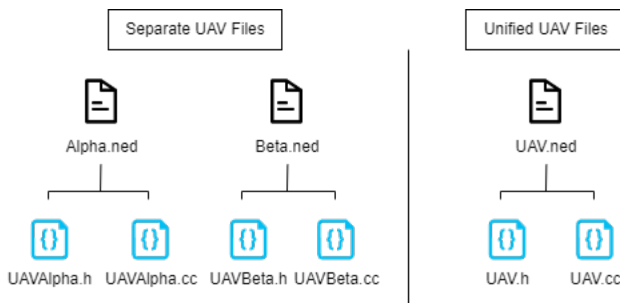

Figure 14. Instances files

found in Gupta et al. (2015). Finally, we detail how to configure the two forms of the property illustrated in Figure 14.

7.6 Limitations and Threats to Validity

Our research adopts a pragmatist worldview (Petersen and Gencel, 2013). In this perspective, our work proposes practical and applicable solutions to the problems faced by Multi-UAV Systems. The developed instances and the architecture and framework consistently provide relevant and useful aspects to meet users' needs to develop Multi-UAV systems. However, they also have some limitations that are discussed in this section.

Regarding **internal validity**, the main assumption is that the Mysterio architecture and framework support the reuse of code and design. The instances provide an initial set of evidence, still limited, that these solutions are providing some assistance as shown in Section 7.4. However, the fact that the authors implemented the instances themselves is a threat to internal validity. We made available all the developed instances and frameworks showing the amount of reused code and, consequently, design decisions. Finally, we understand an evaluation with external practitioners knowledgeable in developing Multi-UAV systems is still required.

The main variables possibly threatening **construct validity** are the main characteristics (Figure 7) and using the amount of reused lines of codes (Table 2). It would be easy to trace those characteristics to concrete components of the instances and to account for components, classes, or design decisions reused for creating the instances. This way, we cannot anticipate additional threats to construct validity.

In terms of **external validity**, we provide instances with multiple characteristics and evolving in complexity. However, relevant aspects are still not evaluated in the current state of our work. All the developed instances that support the assessment of the Mysterio framework and architecture use computer simulations with OMNeT++. Given that, we cannot predict or claim the results will be the same using real-life UAVs. This would demand evaluation encompassing all the challenges related to Sim-to-Real (Chebotar et al., 2019).

Finally, regarding reliability, the instance implementation still relies on the authors. However, all scenarios were based on scenarios from Multi-UAV systems available in the literature. Particularly for the Osborn instance (Section 7.4), we capture even the consensus algorithm from (Kuriki and Namerikawa, 2014), showing the feasibility of using the Mysterio framework to support the implementation of systems with reference algorithms.

8 Conclusions

Due to the importance of Multi-UAV systems, there is a need to support 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 7, we understand that we have achieved an initial architecture and a framework enabling the development of Cooperative Multi-UAV Systems, fostering design and code reuse. The proposed architecture and framework serve as a basis for developers to reuse and develop their cooperative Multi-UAV systems or even evolve the Mysterio framework. Therefore, developers do not need to develop these systems from scratch, saving time and effort and providing simulation capabilities, an essential tool for UAV systems development.

In future work, we intend to assess its use with knowledgeable UAV developers. Thus, we will avoid implementation biases of Mysterio authors in new instances. As we developed our work, we understood the importance of the OM-

NeT++ simulator for developing computer simulations and constructing Multi-UAV systems. We intend to use the framework for future work to develop Multi-UAV systems in other simulators.

Additionally, we plan to improve the next evaluations, including other quality attributes that still need to be evaluated, such as performance and robustness. Given the importance of evaluating the performance of these systems, we plan to work with professionals in vehicular networks to evaluate performance using various network protocols for UAVs. Robustness is another attribute that should be considered since several complex aspects of Multi-UAV can be introduced as potential sources of failure concerning distributed systems characteristics like communication and synchronization and tolerance to hardware faults. Finally, architectural aspects such as testability were not explicitly analyzed even considering we focused on maintainability and reuse. A future question in this direction would be '*How simple is it to test the software with Mysterio compared to not having Mysterio?*' This assessment can be of great relevance in preventing recurring problems in reusable software and helping us identify possible flaws in the reuse of Mysterio.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001 and the Brazilian National Council for Scientific and Technological Development (CNPq), grant #130194/2020-4.

References

Arafat, M. Y. and Moh, S. (2019). Routing protocols for unmanned aerial vehicle networks: A survey. *IEEE Access*, 7:99694–99720.

Asmare, E., Gopalan, A., Sloman, M., Dulay, N., and Lupu, E. (2012). Self-management framework for mobile autonomous systems. *Journal of Network and Systems Management*, 20(2):244–275.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33.

Bandala, A. A., Dadios, E. P., Vicerra, R. R. P., and Lim, L. A. G. (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):745–751.

Bass, L., Clements, P., and Kazman, R. (2012). Software architecture in practice (third edit., p. 624).

Bekmezci, I., Sahingoz, O. K., and Temel, Ş. (2013). Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270.

Briggs, F. (2012). Uav software architecture. In *Infotech@ Aerospace 2012*, page 2539. Researchgate.

Cai, G., Chen, B. M., and Lee, T. H. (2011). *Unmanned rotorcraft systems*. Springer Science & Business Media.

Cavalcante, A. S. N. and De França, B. B. N. (2022). The mysterio framework for developing cooperative multi-uav systems. In *Proceedings of the 16th Brazilian Symposium on Software Components, Architectures, and Reuse*, pages 11–19.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., and Fox, D. (2019). Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE.

Chen, H., Wang, X.-m., and Li, Y. (2009). A survey of autonomous control for uav. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 2, pages 267–271. IEEE.

Daniel, K., Dusza, B., Lewandowski, A., and Wietfeld, C. (2009). Airshield: A system-of-systems muav remote sensing architecture for disaster response. In *2009 3rd Annual IEEE Systems Conference*, pages 196–200. IEEE.

Doherty, P., Granlund, G., Kuchcinski, K., Sandewall, E., Nordberg, K., Skarman, E., and Wiklund, J. (2000). The witas unmanned aerial vehicle project. In *ECAI*, pages 747–755.

Gupta, L., Jain, R., and Vaszkun, G. (2015). Survey of important issues in uav communication networks. *IEEE Communications Surveys & Tutorials*, 18(2):1123–1152.

Hayat, S., Yanmaz, E., and Muzaffar, R. (2016). Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys & Tutorials*, 18(4):2624–2661.

Hong, C. and Shi, D. (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)*, pages 1095–1097. IEEE.

Hrabia, C.-E., Hessler, A., Xu, Y., Brehmer, J., and Albayrak, S. (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*, pages 33–38.

Kekec, T., Ustundag, B. C., Guney, M. A., Yildirim, A., and Unel, M. (2013). A modular software architecture for uavs. In *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*, pages 4037–4042. IEEE.

Krichen, L., Fourati, M., and Fourati, L. C. (2018). Communication architecture for unmanned aerial vehicle system. In *International Conference on Ad-Hoc Networks and Wireless*, pages 213–225. Springer, Springer.

Kuriki, Y. and Namerikawa, T. (2014). Consensus-based cooperative formation control with collision avoidance for a multi-uav system. In *2014 American Control Conference*, pages 2077–2082. IEEE.

Mahmoud, S. Y. M. and Mohamed, N. (2015). Toward a cloud platform for uav resources and services. In *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, pages 23–30. IEEE.

Motlagh, N. H., Taleb, T., and Arouk, O. (2016). Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives. *IEEE Internet of Things Journal*, 3(6):899–922.

Navarro, I. and Matía, F. (2012). An introduction to swarm robotics. *Isrn robotics*, 2013.

Paunicka, J. L., Mendel, B. R., and Corman, D. E. (2005). Open control platform: A software platform supporting advances in uav control technology. *Software-Enabled Control: Information Technology for Dynamical Systems*, pages 39–62.

Petersen, K. and Gencel, C. (2013). Worldviews, research methods, and their relationship to validity in empirical software engineering research. In *2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement*, pages 81–89. IEEE.

Ramos, B. L., Franca, B., Montechi, L., and Colombini, E. (2018). The rocs framework to support the development of autonomous robots. relatório técnico. *Instituto de Computaçao. Universidade Estadual de Campinas (Unicamp), Tech. Rep*.

Ryan, A., Xiao, X., Rathinam, S., Tisdale, J., Zennaro, M., Caveney, D., Sengupta, R., and Hedrick, J. K. (2006). A modular software infrastructure for distributed control of collaborating uavs. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 6455.

Sathyaraj, B. M., Jain, L. C., Finn, A., and Drake, S. (2008). Multiple uavs path planning algorithms: a comparative study. *Fuzzy Optimization and Decision Making*, 7(3):257.

Scherer, J., Yahyanejad, S., Hayat, S., Yanmaz, E., Andre, T., Khan, A., Vukadinovic, V., Bettstetter, C., Hellwagner, H., and Rinner, B. (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*, pages 33–38.

Sharma, V., Sharma, N., and Rehmani, M. H. (2019). Control over skies: Survivability, coverage and mobility laws for hierarchical aerial base stations. *arXiv preprint arXiv:1903.03725*.

Silano, G. and Iannelli, L. (2021). Mat-fly: an educational platform for simulating unmanned aerial vehicles aimed to detect and track moving objects. *IEEE Access*, 9:39333–39343.

Sinsley, G., Long, L., Niessner, A., and Horn, J. (2008). Intelligent systems software for unmanned air vehicles. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, page 871.

Tachinina, O., Lysenko, O., and Alekseeva, I. (2017). Path constructing method of unmanned aerial vehicle. In *2017 IEEE 4th International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, pages 254–258. IEEE.

Tisdale, J., Ryan, A., Zennaro, M., Xiao, X., Caveney, D., Rathinam, S., Hedrick, J. K., and Sengupta, R. (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*, pages 1420–1425. IEEE.

Tisdale, J. P. (2008). *Cooperative sensing and control with unmanned aerial vehicles*. University of California, Berkeley.

Vasudevan, A., Kumar, D. A., and Bhuvaneswari, N. (2016). Precision farming using unmanned aerial and ground vehicles. In *2016 IEEE Technological Innovations in ICT for Agriculture and Rural Development (TIAR)*, pages 146–150. IEEE.

Vincent, P. and Rubin, I. (2004). A framework and analysis for cooperative search using uav swarms. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 79–86.

Yanmaz, E., Yahyanejad, S., Rinner, B., Hellwagner, H., and Bettstetter, C. (2018). Drone networks: Communications, coordination, and sensing. *Ad Hoc Networks*, 68:1–15.

Yu, Q., Cheng, L., Wang, X., Bao, P., and Zhu, Q. (2018). Research on multiple unmanned aerial vehicles area coverage for gas distribution mapping. In *2018 10th International Conference on Modelling, Identification and Control (ICMIC)*, pages 1–5. IEEE.