Adaptive Event Driven Framework for Real Time Multi-Agent Missions

J.A. Bonache-Seco
Computer Science Architecture and Automatics
Complutense University of Madrid
Madrid, Spain
jabonache@ucm.es

J.A. Lopez-Orozco
Computer Science Architecture and Automatics
Complutense University of Madrid
Madrid, Spain
jalopez@ucm.es

Eva Besada Portas
Computer Science Architecture and Automatics
Complutense University of Madrid
Madrid, Spain
ebesada@ucm.es

José L. Risco Martín
Computer Science Architecture and Automatics
Complutense University of Madrid
Madrid, Spain
jlrisco@ucm.es

***Abstract*—A Ground Control Station (GCS) is an essential element to supervise and control autonomous vehicles performing complex missions in real time. In the new era of Internet of Things, where systems are highly connected, these missions demand enormous amounts of computational power to correctly manage the coordination of all the vehicles involved. In this scope, the set of Unmanned Vehicles (UVs) included in the mission must achieve more difficult tasks everyday. As a consequence, the development of a robust, reusable and adaptable GCS framework to allow a single operator to monitor and control a team of heterogeneous agents raises a number of research and engineering challenges. In this paper we introduce an adaptive event-driven framework specially designed for GCSs involved in heterogeneous multi-agent missions that takes advantage of two features: 1) it allows the GCS to add or remove both actual or simulated agents in real time, changing the number or types of monitored agents, and 2) from a software design perspective, the graphical user interface dynamically changes its view in order to minimize operators fatigue and mental workload, facilitating the success of the mission in such complex environments. We also show one of the tests performed with the adaptive framework, where an UV is deployed in a water surface to perform a set of previously planned trajectories and when the trajectories have been successfully completed, a simulated model of another UV will join it to fulfill a leader-follower maneuver.**

***Index Terms*—Event-Driven Architecture, Ground Control Station, Adaptive Graphics User Interface, Mental Workload**

I. Introduction

Modern systems are becoming more complex every day. The new era of Internet of Things and the analysis of large sets of data demand an enormous amount of computational power and the formulation of solutions in real-time. New software, systems engineering paradigms and tools must be developed to tackle the design and implementation of current complex and adaptive systems, which comprise large numbers of heterogeneous components, typically modeled and simulated using complex platforms through the integration of different tools and languages [1].

Traditional distributed simulations and real-time applications that have enormously facilitated the design of these complex systems in the past do not fulfill current constraints related to feasibility, reliability or scalability. Additionally, regarding development platforms and tools, current distributed simulations are supposed to be light, easy to manage, fast, and oriented to services. As a consequence, new simulation paradigms are being designed, as an attempt to include all the large set of requirements into new distributed standards [2].

In this regard, applications of Unmanned Vehicles (UVs) grow every day due to the huge amount of tasks they can carry out successfully on homogeneous missions [3] or teaming with other identical or heterogeneous UVs in heterogeneous configurations [4], [5]. These missions (military or civil) usually imply the development of very specific Ground Control Stations (GCS) for fixed sets of UVs, whether in simulated or realistic environments [6], [7]. However, these frameworks do not adapt well to the evolution of the mission, where some virtual or simulated agents are replaced by real ones, integrating a distributed and real time co-simulation environment.

This paper presents the opportunity to showcase recent research that includes new advances in distributed co-simulation and real-time applications of UV missions, performed in complex scenarios as well. It introduces an adaptive event-driven architecture that allows our GCS to cope with the problems of carrying out missions involving multiple and heterogeneous UVs. To face the problem of UVs variability and co-simulation, our framework combines ideas of event-driven architectures [8] and self-adaptive software [9]. On the one hand, these ideas support asynchronous and loosely coupled software components, required to support different types of sensors and UVs [10], [11]. On the other hand they also provide potential to dynamically modify agents' behavior in response to a self-evaluation or environmental changes [12]–[14], as well as allows us to add or remove certain modules of the architecture like changing the set of UVs available in

each mission.

The remainder of this paper is organized as follows: we show a description of the related work in Section II. The architecture of our proposed framework is detailed in Section III. Section IV shows a complete use case where two UVs are deployed in a water surface to perform a set of previously planned trajectories. Finally, we present conclusions in Section V.

II. Related work

The proliferation of autonomous agents and its technological advances are causing a huge increase in the scope of its applications. They can be found both in military or civil scope, working standalone or teaming with other agents. The nature of these devices and the tasks they are capable of are endless, for example a team of heterogeneous UVs for search & rescue missions, intelligent sensors in the scope of a connected city or home automation using Internet of Things, etc. The features of autonomous agents are improving everyday, so the implementation of software applications flexible enough to monitor/control a set of devices that can be changed at any moment is a continuous challenge for engineers and developers. For this reason, there is an incremental need to find an architectural framework that allows a developer to design a monitoring/controlling software that can cope with the requirements of a set of agents that are going to be dynamically changed because of technological improvements or task reassignment. In the following, we discuss some of the existing approaches developed to tackle the addressed problems and those that have motivated this work.

From a software design perspective, there are several approaches to the architectural design of software platforms oriented to all kind of purposes. Some of them, like Garlan et al. introduces in the Rainbow framework [13], [15] the use of external adaptation mechanisms and an external control loop (a common method in adaptive frameworks) to achieve adaptability. Oreizy et al. [12] describe an infrastructure based on two loops called "Evolution", that manages adaptation over time, and "Adaptation" that cope with reactive changes related to detected events. Other possible approaches are specification languages like Darwin, from Magee et al. [16], a declarative binding language to interconnect dynamic or static structures, or Ponder Policy from Damianou et al. [17], based on roles and event triggered policies suitable for security management, firewalls and other related software. Dashofy et al. [18] designed an architecture for runtime healing systems, that can dynamically auto-repair certain features or modules of its infrastructure.

With respect to the deployment of these systems in real environments, we may find several interesting approaches related to Smart Cities, like the one proposed by Vitaletti et al. [10], that takes advantage of the Event-Driven Architecture features to design infrastructures with loosely coupled event-oriented modules. This allows their infrastructure to monitor simultaneously a huge amount of heterogeneous sensors that broadcast data asynchronously. Hallsteinsen et al. [19] have recently designed a development framework to implement distributed computing systems that allows the clients to be connected to an ubiquitous and dynamic computing environment, for example a connected network for public transport that can be accessed from any operative system to check delays, schedules or even purchase tickets.

Closer to our case, other architectural framework approaches are specifically designed for the implementation of GCSs for UVs. The use of GCSs for UVs is increasing everyday, for civil (leisure, surveillance, etc.) or for military purposes and there is a continuous challenge to implement a flexible, scalable and adaptable GCS that can monitor/control any standalone UV or a team of heterogeneous UVs. There are some works on this direction, like the one from Jovanovic et al. [20], [21], that designed an architecture that is able to control concurrent hard, soft and non real-time tasks, taking advantage of design patterns. They developed an adaptable, reusable GCS to control an Unmanned Aerial Vehicle with some interesting features like the possibility of making a 3D rendering of the terrain to improve situation awareness. Another similar approach can be found in Hong et al. [22], that designed an architecture specifically oriented to real-time software, based on RT-Linux. Amoui et al. [23] developed an approach with runtime adaptivity by means of a model-centric architecture, redirecting control flow to a model interpreter that allows the system to add new program elements or modify the existing ones.

In this paper, we propose the architecture and implementation of a GCS framework. This framework allows an operator (or several operators) to monitor and control a set of heterogeneous autonomous agents. To compose the architecture, we have taken advantage of the previously mentioned features like the adaptive external control loop, the ability to manage simultaneously real-time and non-real-time tasks and the loosely coupled modules implicit in the event driven architecture. All these features, combined, allow us to implement a reusable and scalable GCS framework that may be used to monitor/control a variable team of heterogeneous autonomous agents. Using this framework, the set of agents can dynamically auto-adapt to the environment or changes in the mission objectives. The loosely coupled nature of the event driven architecture gives us the ability to change very easily the amount sensors or UVs involved in the mission without reprogramming an ad-hoc framework for a particular mission stage. Additionally we have added the feature of performing distributed real-time simulations that can be run during a real mission, combined with the actual autonomous agents developed for the experiment. This improves the decision making process, adding or removing agents in real time, depending on the results of the simulations. Moreover, we implemented transparency policies and adaptability [24] in the Graphical User Interface (GUI), to decrease operator's mental workload.

III. Framework architecture

We have designed an adaptive framework architecture based on event-driven philosophy that takes advantage of the ob-

TABLE I
SEARCH WORKS COMPARISON TABLE.

Framework	Oriented to	RT Self Adaptation	Reusable	Extensible	Distributed	Real-Time	RT Simulation	Heter. Agents	Transparency Policies	Multi-purpose
[13], [15]	Architecture	✓	✓	✓	×	✓	×	✓	×	✓
[12]	Architecture	✓	✓	✓	✓	✓	×	✓	×	✓
[18]	Architecture	✓	✓	✓	×	×	×	×	×	✓
[20], [21]	Architecture	×	✓	✓	✓	✓	×	×	×	×
[22]	Architecture	✓	✓	×	×	✓	×	×	×	×
[19]	Architecture	✓	✓	✓	✓	×	×	✓	×	✓
[23]	Architecture	✓	✓	✓	×	×	×	×	×	✓
[10]	Architecture	×	✓	✓	✓	×	×	✓	×	✓
[16]	Specification Language	✓	✓	✓	✓	✓	×	×	×	×
[17]	Specification Language	✓	✓	✓	✓	✓	×	×	×	×
Proposed Approach	Architecture	✓	✓	✓	✓	✓	✓	✓	✓	✓

server pattern for event delivering. It can be used to design infrastructures for almost any kind of multi-agent scopes like GCSs for multiple heterogeneous UVs or any other multi-agent purpose software that may involve monitoring and even controlling a set of heterogeneous devices.

The event-driven architecture helps our framework to decouple agents from sinks and allows data processor modules to receive real-time asynchronous messages, which is very important to deal with the monitoring/control of a set of heterogeneous autonomous agents. That decoupling is also a very important feature for adaptivity as it allows any infrastructure designed following this scheme to change, add or remove any agent (regardless its embedded software or communication protocol) or sink at runtime. This feature makes the software tolerant to changes in the environment or in objectives of the current task. Additionally, the decoupling enhances real-time distributed simulation permitting a remote computer to generate data and granting it to join the system at runtime.

As we can see in Fig. 1, the framework is formed by three blocks of modules: 1) agents, that are the autonomous modules that carry out the tasks, sense the environment and generate data for the system, 2) channels, that are the mechanism used to transfer an event from agents to sinks, and 3) sinks that are the event (data) processors responsible of identifying each event and react to it with the appropriate actions. Next, we explain each block in detail.

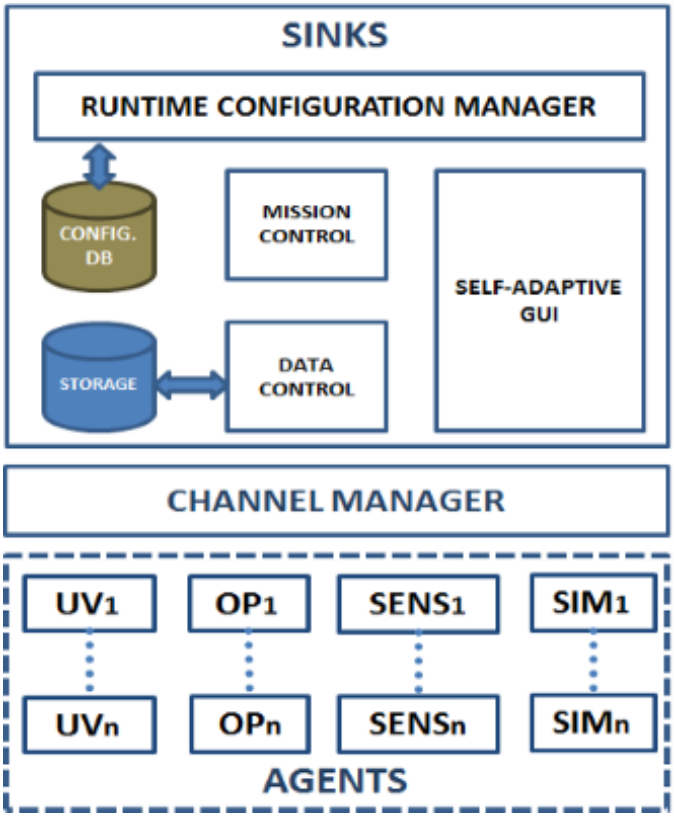

Fig. 1. Architecture of the proposed adaptive event-driven GCS framework

A. Agents

In our event-driven architecture, an agent is a module that senses something and generates an event. Our framework allows any autonomous connected device to join the system at runtime. It is possible thanks to a communication layer added to the device's embedded hardware (or in an auxiliary micro-computer like a Raspberry Pi, Intel Edison, etc., if necessary) and a previously defined XML configuration file that tells the system how to configure and communicate with the agent. In the following, the four main types of agents considered are described: Unmanned Vehicles, External Sensors, Simulation Engines and Operators.

The first type of agent considered is the Unmanned Vehicle (Surface, Aerial, Ground, etc.). These are a special kind of agents that must send data (events) through the channel manager and receive certain commands or path planning from the planner/commander displayed on the operator's adaptive GUI. UV agents generate data that is encapsulated in two kind of events: (a) *data events*, that encapsulate vehicle telemetry, mission task status, vehicle sensor data, etc., and (b) *alarm events*, that sends an urgent message with any critical infor-

mation to be displayed to the operator and processed by the mission control module that will be explained in the sinks Section.

The second type of agent is the external sensor, that create events related to any kind of sensed information from the environment, a specific data (climate, pressure, motion), a surveillance system, etc.

The third type of agent is the simulation engine. These are remote computers with the adaptive framework communications layer and a programmed model of a certain kind of agent (an UV navigation model, for example), that allows the system to perform distributed real-time simulations. Data are generated in the remote computer and sent through the channel manager to the sinks with the same mechanisms as a real agent, so for the other components of the framework, the simulation is a transparent operation.

In the scope of this framework, an operator is considered as an autonomous agent because he or she can produce a certain type of events, named *operator requests*. These requests are used by an operator to ask the system for information like agent status, task status, etc. There is also an special operator request, named *Infrastructure Change Request (ICR)*, which is one of the most important features in the adaptive event-driven CGS framework. An ICR is a message used to synchronize all the parts involved in a critical infrastructure change like an agent requesting to join the system or an operator delegating tasks to another one. When an agent connects to the system via the channel manager, an ICR is generated by the control manager and received by the appropriate sink, in this case, the adaptive GUI. The interface shows a message to the main operator that must give permission to perform the infrastructural changes. The adaptive mechanisms are launched by the runtime configuration manager based on the data stored in the configuration database, that knows how to adapt all of the modules related to this specific kind of agent. A task delegation generate an ICR requested by an operator (OP1) that is overwhelmed by the amount of tasks that he/she has to accomplish. OP1 must choose another free Operator (OP2) that gives consent to the task delegation. Then the graphic elements of the delegated task appear on the GUI of OP2 that must push the "Ready" button to show to the system that he/she is ready. At this point, the graphic elements of the task disappear from the OP1 GUI releasing him/her of this particular task.

B. Channels

Channels are the software mechanisms used to transfer an event from agents to sinks. In our adaptive event-driven framework, there is only one multipurpose channel named Channel Manager (CM) that perform all channel related tasks.

The connection between distributed modules of the framework are TCP/IP socket communications to ensure the compatibility between all kind of computers and operative systems, and take advantage of the Observer pattern to broadcast messages from agents to sinks through CM, so CM is structured as the Subject and the sinks as the Observers, as Fig. 2 depicts.

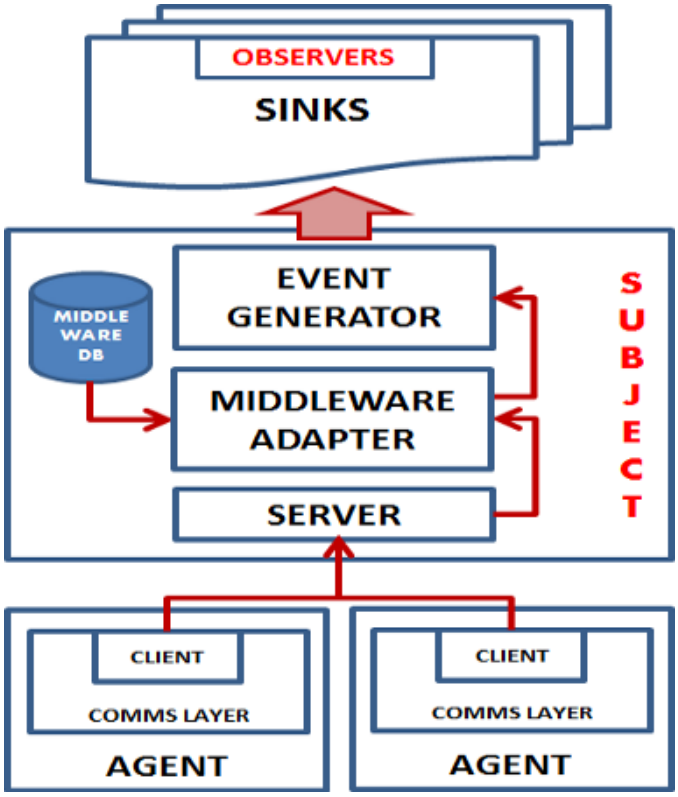

Fig. 2. Communications Data Flow

The Subject is designed as a multi-layer module with a server, middleware adapters and an event generator.

The *server* is the layer that manages connection requests and establishes a robust and secure communication channel between remote agents and sinks. The *middleware adapters* are the modules that decode different protocols from the autonomous agents helping to decouple the "producer" modules from "consumers" in the framework. The keys to decode all the protocols to be used by the agents are previously stored in the *middleware database*. When the message is decoded, data is delivered to the *event generator*, where it is encapsulated in an event object and the subject notifies all the observers that the event has been created, so they can receive it and react to the encapsulated data.

To illustrate how the data are processed by CM, an example of data flow can be observed in Fig. 2. First of all, an agent connects to the software infrastructure through a request sent by the socket client of its communications layer. The server validates the HandShake and then opens a socket connection between them. From this point, CM receives the agent messages that are transferred to the corresponding middleware adapter that decodes it and deliver the data to the event generator. In this layer, data are encapsulated in an event and a notification is sent to all the Observers. The corresponding sink (Observer) receives the message and reacts to it with the appropriate mechanisms.

C. Sinks

Sinks or event processors are the modules responsible of identifying an event and reacting to it with the appropriate mechanisms. As previously described, sinks play the role of observers in the Observer pattern. This provides to our framework modules the ability to be very easily developed because the only requirement is to implement the "update" method in the Observer pattern. This feature allows the proposed framework to be very flexible and reusable. Additionally, it can be easily adapted to any change in the environment or mission objectives by adding, changing or removing modules at runtime. In our proposed framework, there are four main types of sinks: data control, mission control, runtime configuration manager, and self adaptive GUI.

The *data control* module is the sink that manages the data received from the agents. It filters relevant data to the framework and stores them in a database to be extracted, processed and analyzed in future experiments and improvements.

The *mission control* is the module that processes and reacts to events related with the tasks to be fulfilled by the Agents or the sensed data monitored by the infrastructure. For example, regarding an infrastructure designed to be a GCS for multiple heterogeneus UVs, it is the module in charge of triggering reactive alarms (e.g. launch an evasive maneuver if an obstacle is detected or reducing speed if a high speed alarm is received). This kind of automated mechanisms help the system to reduce mental workload to the operator.

The *runtime configuration manager* is the module that deploys the selected changes in the infrastructure, that is, adds, removes or changes modules at runtime to adapt it to a new situation. When the system detects a new requirement that cannot be fulfilled or an ICR is received, the runtime configuration manager starts the mechanisms to adapt the software infrastructure. The rules to implement the changes in the infrastructure are stored in the *configuration database*, that can read some additional data from an XML configuration file if a new agent is going to join the infrastructure. When operators give permission and the system reaches a non-critical situation, the runtime configuration manager performs the required changes.

The *self-adaptive GUI* is a very important module of the proposed framework because is the responsible of showing all relevant data to the operator. Data must be displayed in the more ergonomic and easier way possible in order to reduce operator's mental workload. This sink is configured as an adaptive infrastructure itself using the same philosophy followed in the adaptive event-driven framework. The self-adaptive GUI receives data and reconfigures at runtime if necessary to show all the information needed and to ease the work of the operator, that is able to monitor/control multiple autonomous agents simultaneously. This is possible thanks to two implemented mechanisms: adaptation and transparency.

Fig. 3 shows the architecture of the self-adaptive GUI, which is divided in four blocks: an adaptive runtime engine, a rules and profiles database, a controller, and a view. The

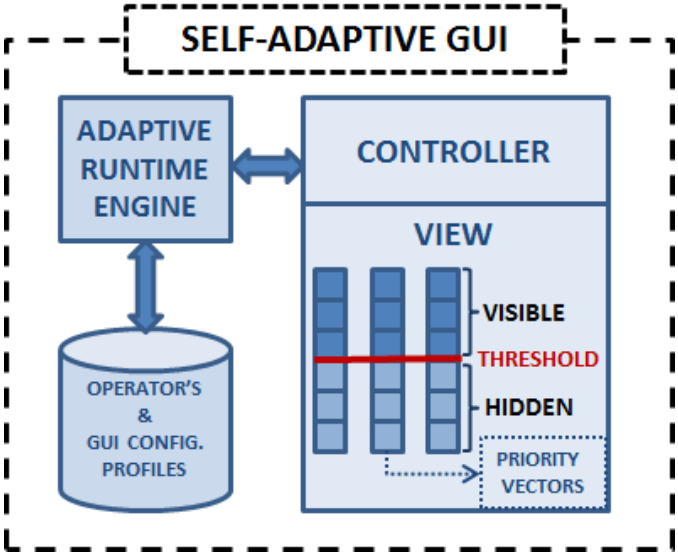

Fig. 3. Architecture of the Self-Adaptive GUI

adaptive runtime engine is the module that checks if a change is necessary and deploys it using the infrastructure rules and the operator's profile stored in a database. The changes are operated by the controller and may be caused by a new agent joining the infrastructure, so new data must be monitored, by an operator request, or by certain measurements like operator's mental workload. The first way to reduce mental workload are the adaptation mechanisms, that move certain graphic elements to adjust to the priorities of the task goals and the operator ergonomic preferences. The second one are the transparency policies. When too many graphic elements are displayed, the operator takes more time to find the relevant ones, so we implemented priority vectors where elements with a priority lower than a threshold are not displayed. These mechanism hide data that is considered non-relevant for a certain task. When the set of tasks changes, the priority for each graphic element and the threshold are recalculated by the adaptive runtime engine to decide which ones are relevant and which ones can be hidden.

IV. Use Case

The adaptive event-driven framework has been tested on several field experiments in different scopes and involving heterogeneous teams of autonomous agents. The framework allows the developer to easily deploy an adaptive infrastructure to monitor or control several autonomous agents that can be reassigned if the tasks or the environment of the mission changes.

In this section we describe one of the use cases conducted in the scope of the SALACOM project, where an Unmanned Surface Vehicle (USV1) is deployed in a water surface to perform a set of previously planned trajectories (lemniscata, circumference, predefined waypoint navigation, etc.) [25]. When the trajectories have been successfully completed, USV1 will reach a certain waypoint where a simulated

model of another Unmanned Surface Vehicle (USV2) will join it to fulfill a leader-follower maneuver. The objectives of this experiment are 1) verify that USV1 correctly fulfills the preplanned trajectories, 2) verify that USV2 correctly follows the leader ship USV1 in a leader-follower maneuver, and 3) test the adaptive infrastructure in the scope of the experiment and check if the transition between the scenarios with one and two autonomous agents is feasible. Additionally, there are several side objectives like testing the collision alarm that is raised when USV1 and USV2 are too close, confirm that only one operator can cope with the monitoring and control of both vehicles and, above all, completing the first set of collaborative maneuvers between USV1 and USV2 with the advantages of the real time distributed simulation. This feature of our adaptive framework allows us to perform the maneuvers without danger of a collision between USVs, that could be fatal for the hardware if one or both of the USVs sinks.

The experiment, as seen in Fig.4, is configured with two USVs, as scale models of real ships designed by the Canal de Experiencias Hidrodinámicas de El Pardo (CEHIPAR). USV1 (leader) is four meters length and USV2 (follower) five meters length. As we explained earlier, a USV2 computer model is only going to be used inside a real-time simulation engine. Both USVs have a Beckoff built-in computer, model C6920, with the TwinCAT software system, a real time system with embedded Windows OS. USV1 has a VectorNav VN-200 Inertial Measurement Unit (IMU) with embedded GPS and USV2 has CodaOctopus IMU-F180 IMU GPS/INS. The communication system is a Wi-Fi wireless network formed by the antenna Ubiquiti NanoStation M2 as access point (1.5 to 2 Km^3 range) and one Ubiquiti PicoStation M in each USV. Depending on the experiment, the USVs can carry another sensors like radar, depth probe, etc.

To begin with the field experiment, the USV1 is deployed for the phase 1: standalone trajectory and path planning tests. The adaptive framework configures itself with a simple structure formed only by one GUI, the channel manager and only one autonomous agent, the USV1 (see Fig. 5). The GUI includes a planner, a controller and all the graphic elements considered to let one single operator to monitor and control the USV (position on the map, waypoints, telemetry data, graphic charts, etc.). The channel manager is the module responsible of setting up the TCP/IP socket connection between an autonomous agent, USV1 in this case, and the sinks (GUI, data management module, mission management module, etc.). When the autonomous agent is turned on, its framework's communications layer makes a communication request to the channel manager, that sets up a TCP client that connects to the channel manager server. Once the communication channel is set up, the channel manager maintains a secure and robust communication channel between agents and sinks. The agent sends messages through the channel manager that processes them and encapsulates the information received into an event structure, that is understandable by any sink of the framework. Events are broadcasted through the infrastructure thank to the Observer pattern described in Section III and their information

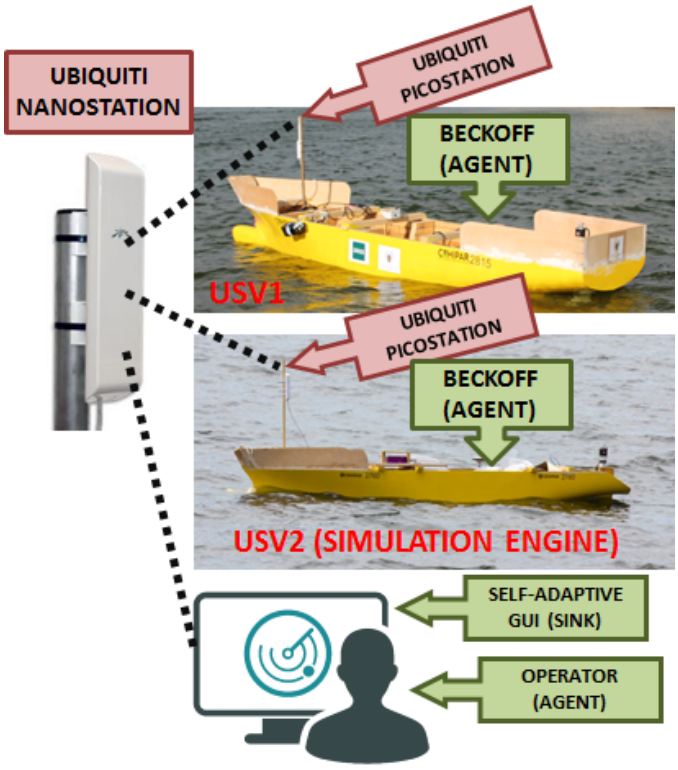

Fig. 4. Field Experiment Hardware Configuration

processed by the appropriate sink or sinks. During phase 1, the operator conducted several trajectories using the planner and the controller included in the GUI. In Fig. 6 we can observe one of the parametric trajectories successfully completed, the lemniscata. The red line represents planned trajectory and blue line represents the trajectory that the USV1 carried out.

When phase 1 is finished and the standalone trajectories completed, the operator introduces a prefixed waypoint in the commander and when USV1 reaches it, the phase 2 begins. At this moment, USV2 is turned on and its communication layer sends the communication request to the channel manager that set up a communication channel as seen on phase 1 with USV1. Once the communications channel is established, that is, USV2 is requesting entering the mission, the adaptive framework sends an ICR that needs to be validated by the operator. If the operator gives permission, the runtime manager accesses to the stored rules of change and set the necessary modules to the adaptive framework. In this case, the configurator added the necessary graphic elements to the GUI (USV2 icon on the map, telemetry data and trajectory graphs USV1-USV2) and the data and mission controllers for the management of USV2 data storage and mission objective checks. The new configuration can be seen on Fig 7.

During phase 2 we tested all the subjects previously planned for the field experiment. First of all, we proved successfully the leader-follower maneuver using USV1, physically deployed on the water and USV2, that simulated in its embedded computer a real-time model of itself and its controller, allowing us to

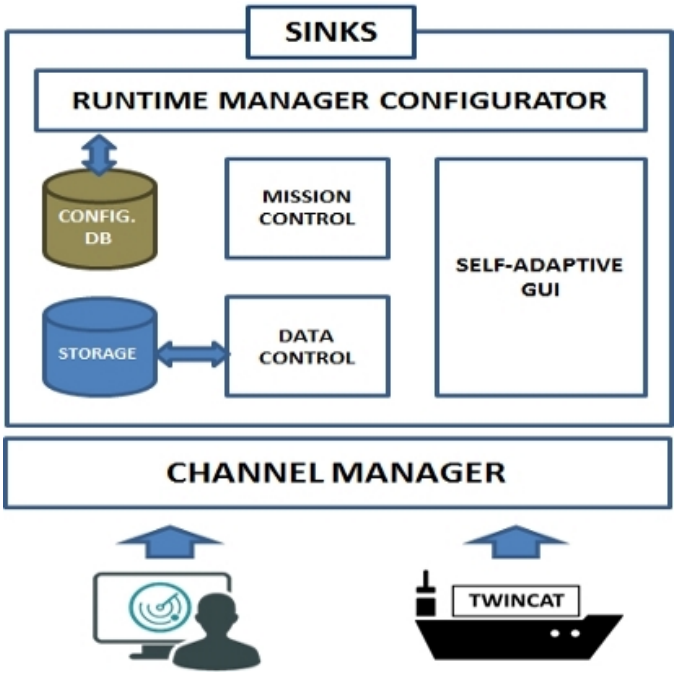

Fig. 5. Use Case Infrastructure - Phase 1

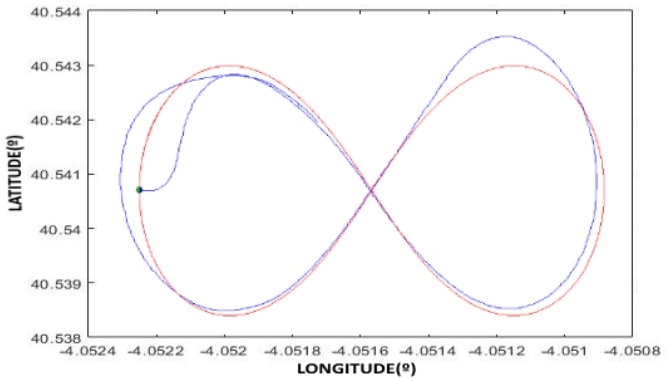

Fig. 6. Trajectory completed during Phase 1

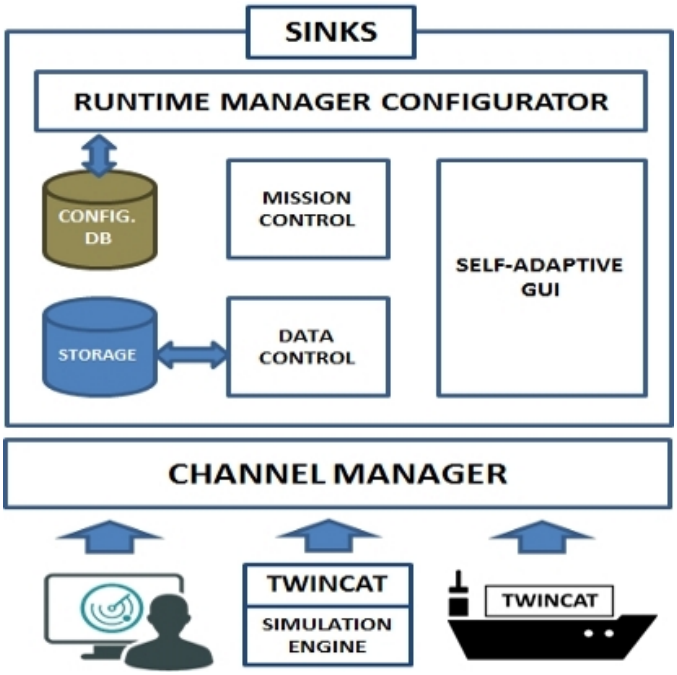

Fig. 7. Use Case Infrastructure - Phase 2

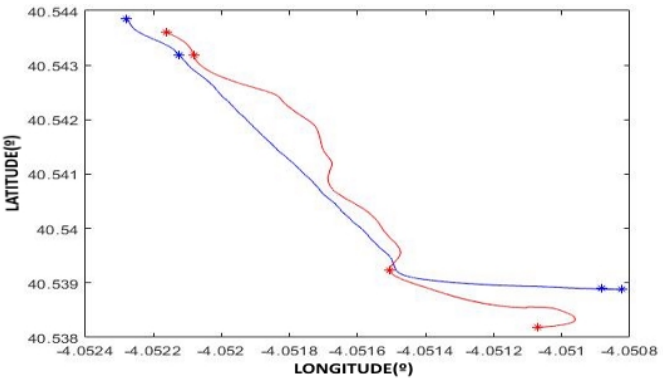

Fig. 8. Trajectory completed during Phase 2

conduct a leader-follower maneuver without the dangers of two USVs physically deployed on the water. Moreover, this simulation helped us to determine that the adaptive framework is able to reconfigure its structure at runtime without any error, as the mechanisms used by the runtime manager are exactly the same that it would use in a mission with USV2 physically deployed. Using USV2 as a Simulation Engine, we were capable of testing a set of predefined alarms (high speed, collision danger) without endanger any hardware. In this phase we successfully completed a leader-follower maneuver as seen on Fig. 8, where the blue line represents the leader ship, USV1, physically deployed for the experiment, whereas the red line represents the follower ship, USV2, where data were real-time generated in a remote Beckoff computer.

Additionally, when phase 2 was almost concluded, we checked the mechanism of task delegation. For this, we set up a second GUI on another computer that was going to be used as a second GCS, managed by another operator. We connected it to the adaptive framework, that detected it automatically and configured its data reception as sink, beginning to monitor the predefined vehicles in its XML configuration document. Then the operator 1 selected task delegation on the operator 2 and selected the vehicle which tasks were going to be delegated. This sent the corresponding ICR and activated the delegation mechanisms that prevented operator 2 to give his consent. When the operator 2 gave his permission, the system received confirmation an selected the new structure, as seen in Fig. 9

V. Conclusion

This paper has described the design an implementation of a distributed GCS framework. We first have described the architecture of the framework, which is mainly formed by three layers: agents, channels and sinks. The agents layer

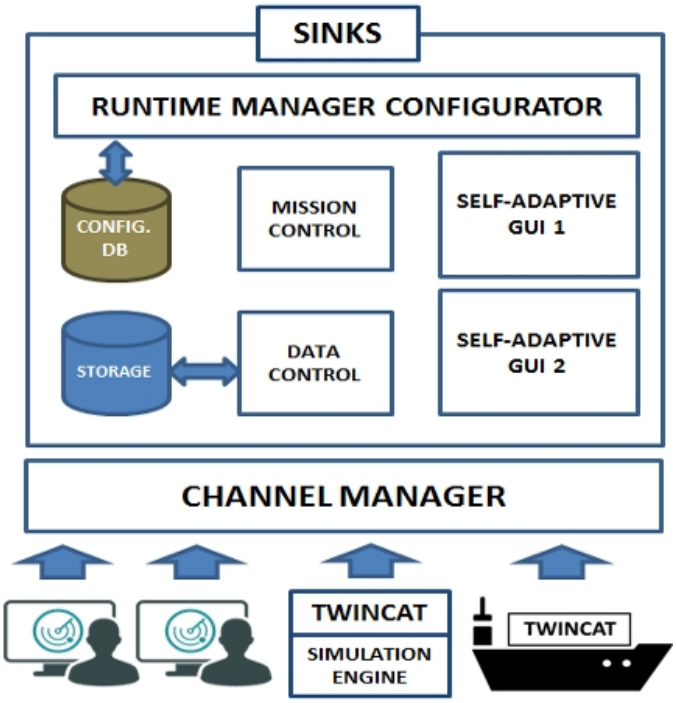

Fig. 9. Use Case Infrastructure with 2 SA GUIs

include every module able to sense something and generate events, such as UVs, external sensors, simulation engines, or operators. The channels layer are the software mechanisms used to transfer events from agents to sinks. Finally, the sinks layer are responsible of identifying an event and reacting to it with appropriate mechanisms.

Our GCS framework has been tested with several use cases. This paper has shown one of them, where an USV is deployed in a water surface to perform a set of predefined trajectories. When the trajectories are completed, a second and simulated USV joins the first one to fulfill a leader-follower maneuver. The possibility of including simulated models in the experiments is transparent to all the other agents and saves potential risks and damages in trials with several UVs. Finally, the test included the addition of a second GCS to the framework, allowing the inclusion of a second operator in real time.

REFERENCES

[1] S. Mittal and J. L. Risco-Martín, *Guide to Simulation-Based Disciplines: Advancing Our Computational Future*. Springer, 2017, ch. Simulation-based Complex Adaptive Systems, pp. 127–151.

[2] ——, "DEVSML 3.0: Incorporating Docker and Microservices for Rapid Deployment of DEVS Farm in Cloud Environment," in *Proceedings of the 2017 Spring Simulation Multionference (SpringSim 2017)*, 2017.

[3] R. Sutton, S. Sharma, and T. Xao, "Adaptive navigation systems for an unmanned surface vehicle," *Journal of Marine Engineering and Technology*, vol. 10, pp. 3–20, 2011.

[4] R. R. Murphy, E. Steimle, C. Griffin, C. Cullins, M. Hall, and K. Pratt, "Cooperative use of unmanned sea surface and micro aerial vehicles at hurricane Wilma," *Journal of Field Robotics*, vol. 25, pp. 164–180, 2008.

[5] M. Lindemuth, R. Murphy, E. Steimle, W. Armitage, K. Dreger, T. Elliot, M. Hall, D. Kalyadin, J. Kramer, M. Palankar, K. Pratt, and C. Griffin, "Sea robot-assisted inspection," *IEEE Robotics Automation Magazine*, vol. 18, no. 2, pp. 96–107, June 2011.

[6] J. Heo, S. Kim, and Y. J. Kwon, "Design of ground control station for operation of multiple combat entities," *Journal of Computer and Communications*, vol. 4, no. 05, pp. 66–71, 2016.

[7] A. Bürkle, F. Segor, M. Kollmann, and R. Schönbein, "Universal ground control station for heterogeneous sensors," *Journal On Advances in Telecommunications*, vol. 3, pp. 152–161, 2011.

[8] B. M. Michelson, "Event-driven architecture overview," *Patricia Seybold Group Research Service (2006)*, 2006.

[9] R. Laddaga and P. Robertson, "Self adaptive software: A position paper," in *SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, vol. 31. Citeseer, 2004, p. 19.

[10] L. Filipponi, A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci, "Smart city: An event driven architecture for monitoring public spaces with heterogeneous sensors," *Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM)*, 2010.

[11] O. Etzion, "Towards an event-driven architecture: An infrastructure for event processing position paper," *Lecture Notes in Computer Science book series (LNCS, volume 3791)*, 2005.

[12] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems and Their Applications*, vol. 14, no. 3, pp. 54–62, 1999.

[13] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer (IEEE Computer Society)*, vol. 7 (Issue: 10), pp. 48–54, October 2004.

[14] P. Oreizy, M. M. Gorlick, and R. N. Taylor, "An architecture-based approach to self-adaptive software," *Intelligent Systems and their applications*, vol. 14, no. 3, 1999.

[15] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," *SEAMS 06 Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pp. 2–8, May 2006.

[16] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, "Specifying distributed software architectures," *Software Engineering - ESEC 95 Lecture Notes in Computer Science*, vol. 989, pp. 137–153, 1995.

[17] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," *Policies for Distributed Systems and Networks series Lecture Notes in Computer Science*, vol. 1995, pp. 18–38, February 2001.

[18] R. N. T. E. M. Dashofy, A. van der Hoek, "Towards architecture-based self-healing systems," *WOSS '02 Proceedings of the first workshop on Self-healing systems*, 2002.

[19] S.Hallsteinsen, K.Geihs, N.Paspallis, F.Eliassen, G.Horn, J.Lorenzo, A.Mamelli, and G.A.Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," *Journal of Systems and Software*, 2012.

[20] D. S. Mladan Jovanovic, "Software architecture for ground control station for unmanned aerial vehicle," *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2010.

[21] Z. J. Mladan Jovanovic, Dusan Starcevic, "Improving design of ground control station for unmanned aerial vehicle: Borrowing from design patterns," *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, 2008.

[22] W. E. Hong, J. S. Lee, L. Rai, and S. J. Kang, "Rt-linux based hard real-time software architecture for unmanned autonomous helicopters," *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, 2005.

[23] M. Amoui, M. Derakhshanmanesh, J. Ebert, and L. Tahvildaria, "Achieving dynamic adaptation via management and interpretation of runtime models," *Journal of Systems and Software*, 2012.

[24] J. E. Mercado, M. A. Rupp, J. Y. C. Chen, M. J. Barnes, D. Barber, and K. Procci, "Intelligent agent transparency in human-agent teaming for multi-UxV management," *Human Factors*, vol. 58, no. 3, pp. 401–415, 2016.

[25] J. M. de la Cruz, J. A. Lopez-Orozco, E. Besada-Portas, and J. Aranda-Almansa, "A streamlined nonlinear path following kinematic controller," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6394–6401.