ART-GCS : AN ADAPTIVE REAL-TIME MULTI-AGENT GROUND CONTROL STATION

Juan A. Bonache-Seco
José A. Lopez-Orozco
Eva Besada-Portas
José L. Risco-Martín

Department of Computer Architecture and Automation
Complutense University of Madrid
Plaza Ciencias 1, Madrid, Spain
{jabonache,jalopez,ebesada,jlrisco}@ucm.es

ABSTRACT

Ground Control Stations (GCS) are essential tools to monitor and command real-world complex missions involving Unmanned Vehicles (UVs). As the number and types of UVs in the mission grows, implementing a robust and adaptable GCS, capable of simplifying and reducing operator' interactions and mental workloads, becomes an engineering challenge. To address it, this paper presents a new Adaptive-Real-Time (ART)-GCS that 1) allows to monitor and control a runtime changing number of heterogeneous UVs, 2) adapt its GUI to the mission requirements and operators workload to minimize their fatigue and stress, and 3) provide support to experiments with actual and simulated UVs. To show its benefits in real-world missions, this paper presents a field experiment where, for safety reasons, a simulated unmanned aerial vehicle has to find an oil-spill that must be enclosed by a containment boom dragged by two real unmanned surface vehicles.

Keywords: Ground Control Station, Adaptive Graphics User Interface, Mental Workload, Complexity, Transparency

1 INTRODUCTION

Nowadays, monitoring and control systems must be designed to cope with greater sets of heterogeneous agents, which are needed to accomplish tasks or missions inconceivable a few years ago. Occasionally, part of the agents must be simulated too, in order to reduce potential risks for their hardware in critical situations (e.g. to test the feasibility of the whole system in adverse environment conditions when a new agent is added to the mission at runtime). Besides, autonomous agents constantly evolve including new features to fulfill their standalone or teaming tasks more efficiently, even in changing environments. For this reason, new software engineering paradigms must be developed to support the implementation of flexible and adaptive complex systems that tend to be distributed, to mix real-time and non real-time applications and, usually, to integrate different tools, programming languages and operative systems. Also, they need to be capable of processing huge sets of data and ergonomically displaying them to the operators in charge of the mission.

Traditional approaches do not meet all the requirements needed for systems with the previous characteristics and are usually designed ad hoc for very specific missions (i.e. for a fixed agent/team of agents that carries/carry out a certain task in a particular environment). These traditional developments, either for systems with agents working standalone (Sutton et al. 2011, Ribas et al. 2012, Patterson et al. 2013) or teaming with sets of homogeneous or heterogeneous agents (Murphy et al. 2008, Bürkle et al. 2011, Lindemuth et al. 2011, Heo et al. 2016), usually result in non-reusable software, very difficult to adapt to changes in the environment, the agents or their tasks. Moreover, monitoring huge sets of data may cause high rates of stress and mental workload to operators or make them neglect some of their tasks.

To overcome the limitation of existing software, this paper presents ART-GCS, a new Adaptive Real-Time Ground Control Station that allows operators to successfully monitor and command complex missions involving multiple heterogeneous Unmanned Vehicles (UVs). It is built over the adaptive event driven framework outlined in (Bonache-Seco et al. 2018), which, combining ideas of event-driven architectures (Michelson 2006) and self-adaptive software (Laddaga and Robertson 2004), provides the infrastructure needed for dynamically modifying agents' behavior or adding/removing certain software components in response to self-evaluation or environmental changes (Oreizy et al. 1999, Garlan et al. 2004). In addition, ART-GCS aims to reduce operator's mental workload, time of reaction and stress by implementing transparency policies on the GCS' Graphics User Interface (GUI), which are based on the idea of hiding non-relevant information at certain points of the mission, adjusting the amount of available data to help operators to focus on the most important information (Chen et al. 2014, Mercado et al. 2016). ART-GCS also adapts the organization of its GUI to the operator preferences. Moreover, it is itself a perfect platform to test: 1) Artificial Intelligence techniques for decreasing operators mental workload, and 2) the capability of handling in the same mission heterogeneous physical and real-time-simulated agents to reduce hardware risks during the initial developments of the missions.

The remainder of this paper is organized as follows. A review of related work that highlights the most relevant features of the new GCS is presented in Section 2. The underlying architectural design of ART-GCS, detailed in (Bonache-Seco et al. 2018), is sketched in Section 3. The mechanisms that support the adaptability of the GUI of ART-GCS, one of the main contributions of the paper, are detailed in Section 4. Section 5 illustrates the benefits of ART-GCS, focusing on the adapability of its GUI, while describing a complete use case where two Unmanned Surface Vehicles (USVs) and an Unmanned Aerial Vehicle (UAV) are deployed in a mission to carry out a cooperative task. Finally, Section 6 draws the main conclusions.

2 RELATED WORK

At present, the technological advances in autonomous agents, sensors and Internet of Things open new scopes for multiagent missions where complex adaptive systems are required to efficiently handle the existing diversity of devices, tasks and environments. The GCSs of these systems should aim to be flexible, reusable and extensible in order to fulfill the requirements of monitoring/controlling a set of devices that can be modified at any time in a changing environment. Moreover, the increasing number of agents involved in nowadays missions often increments the number of operators needed to handle them successfully. And the biggest the number of devices to be controlled, the highest the mental workload of stressed-out and neglecting operators. All these facts introduce new challenges for GCS engineers and developers, who need to use frameworks that let them handle all the GCS requirements and who must implement friendly GUIs that reduce the number of operators in the mission by helping them to observe easily the most relevant data.

In the following, we first discuss some of the state of the art approaches developed for addressing the problems previously mentioned, focusing our attention on those works that motivated ours and on their following key features: adaptation by means of external control loops, heterogeneous software and hardware integration by means of the loosely coupled philosophy of event driven architectures, and relevant data identification and display by means of transparency and adaptation policies. Afterwards, some existing GCSs and other adaptive software are analyzed and their limitations highlighted.

The first feature, applied in several *adaptive* software platforms, consists of including in their architectural design external control loops that check, continuously, if different types of changes occur, to adapt, if necessary, the software behavior at runtime. More in detail, Rainbow by (Garlan et al. 2004, Cheng et al. 2006) follows this strategy by implementing a single external control loop to identify system changes, while the self-adaptive approach by (Oreizy et al. 1999) applies an "Evolution Loop" and an "Adaptation Loop" to respectively adapt the software over time and after detecting certain events. The benefit in both cases, which the underlying framework of ART-GCS presented in (Bonache-Seco et al. 2018) also enjoys, is the development of self-adaptive pieces of software (GCSs in our case).

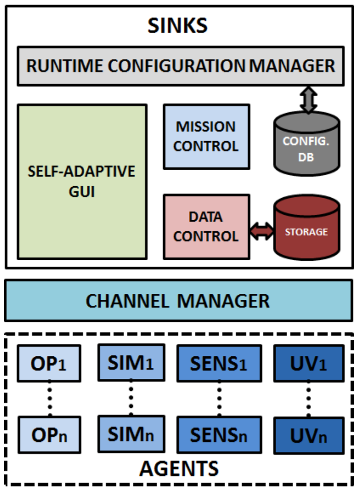
Another key aspect, especially relevant for the software deployment in real-world complex environments involving multiple heterogeneous actors, consists in taking advantage of the features of *Event-Driven Architectures* (EDAs) to design distributed infrastructures with loosely coupled modules, capable of managing real-time connections independent on hardware or software bindings (Etzion 2005). An interesting example of this idea, related to Smart Cities and developed to simultaneously monitor a huge amount of heterogeneous sensors that broadcast data asynchronously, is presented in (Filipponi et al. 2010). A similar strategy is followed by (Hallsteinsen et al. 2012) for distributed computing systems that allow clients with different hardware and operative systems to connect to an ubiquitous and dynamic computing environment formed by a wireless network. The benefits for our case are obvious: the underlying EDA of our GCS allows it to handle at runtime a changing set of heterogeneous UVs (real or simulated), sensors and operators.

Last but not least, the use of *transparency policies* is also really interesting to decide which data have to be automatically displayed (if they are considered relevant) or hidden (if they are not) on the GUI of a GCS, as it helps operators to focus on the most relevant information, and to reduce their levels of stress and mental workload (Mercado et al. 2016). Hence, these types of policies are incorporated in ART-GCS to create an ergonomic and operator-friendly GUI, with the purpose of allowing a single operator to monitor and control several UVs simultaneously.

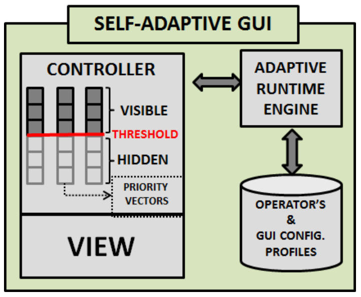
Existing GCSs, which allow an operator (or, more commonly, a set of operators) to remotely monitor and control one or more UVs during complex missions, are often not-so-versatile but quite-effective solutions, designed either for civil (especially for leisure UAVs) and military purposes. The following two general-purpose GCSs stand out. (Jovanovic et al. 2008, Jovanovic and Starcevic 2010) propose an architecture specifically designed to develop distributed and reusable single-UAV GCSs that supports the integration of concurrent hard, soft and non real-time tasks but that does not self-adapt, handle heterogeneous physical or simulated vehicles, or reduce operators mental workload. (Hong et al. 2005) introduce an architecture specifically oriented to real-time GCSs based on RT-Linux that adapts at runtime, but that doesn't support distributed computing, simulations or the inclusion of heterogeneous agents.

Two additional pieces of software that, in spite of not being related with GCSs, incorporate interesting adaptive mechanisms should be mention too. On one hand, the self-adaptive, reusable and extensible model-centric architecture by (Amoui et al. 2012) supports runtime adaptivity by redirecting the control flow to a model interpreter that allows a third module to change, add or remove some program elements. On the other one, the self-adaptive, reusable and extensible architectural runtime healing system by (E. M. Dashofy 2002) dynamically auto-repairs certain modules of its infrastructure to improve their performance. None of them supports distributed computing, real-time applications or simulations, or the integration of heterogeneous autonomous agents.

Finally, after highlighting the benefits and drawbacks of existing GCSs and pieces of adaptive software, it is worth summarizing the advantages of ART-GCS, which is a reusable and scalable tool especially intended to help a single operator (or several if it is necessary) to monitor and control a team of heterogeneous autonomous agents. On the one hand, its underlying EDA framework allows to swap, add or remove the (physical or simulated) UVs engaged in a mission at runtime (without having to reprogram ad-hoc internal



(a) General Overview.



(b) GUI Architecture.

Figure 1: ART-GCS Adaptive Event Driven Architecture.

software modules for new types of vehicles). On the other, its adaptable transparency-policy-based GUI decreases operators stress and mental workload, and increases their decision making and vehicle handling capabilities.

3 ART-GCS ARCHITECTURE

This section sketches the software architecture of ART-GCS, which is supported by the Adaptive Event-Driven Architecture (AEDA) detailed in (Bonache-Seco et al. 2018). We use AEDA for ART-GCS since it is a framework that allows developers to easily design and implement complex adaptive (runtime changeable) infrastructures for monitoring and controlling sets of heterogeneous agents (working standalone or teaming to perform more complicated missions), due to the loosely coupled nature of its modules (agents - sinks), its uniform asynchronous distributed message management, and its distributed real-time computing and simulation capabilities.

The ART-GCS architecture is formed by the three modular blocks represented in Figure 1a and consistent of the Agents (autonomous modules that perform the mission tasks, sense the environment and gather the data), the Channel Manager (in charge of the communications and message management) and the Sinks (data processors that identify the received events and their data, and react to them with the appropriate actions). Although detailed information of their functionality can be found in (Bonache-Seco et al. 2018), their most relevant aspects are next indicated to help readers to understand the underlying mechanisms that support the behavior of the self-adaptive GUI presented in Section 4.

Agents are the actors involved in the mission. They have a communication layer implemented on their embedded hardware (or in an auxiliary micro-computer) to send (and encapsulate in an event) the information they provide/gather from the current task. They can take advantage of the loosely coupled nature of EDA to join the mission at runtime (as allowed in a previously defined XML configuration file). In particular, there are four types of agents: a) UVs that send data and receive commands through the Channel Manager, b) Sensors that gather environment information to be monitored in the Sinks, c) Simulation Engines that implement UVs or Sensor models and generate their corresponding real-time simulation data to be treated indistinctly of the real ones, and d) Operators that are considered agents because they produce certain types of events called operator requests. Finally, as (real) UVs and Simulation Engines are similarly treated within ART-GCS, there is no need to set-up special mechanisms (outside of the Channel Manager explained below) to exchange data among them.

The **Channel Manager** is the module used to deliver the events (with encapsulated data) from Agents to Sinks. It is formed by thee elements: a) Server that manages all the connection requests and establishes a secure TCP/IP socket communications channel (compatible with all computers and operative systems) between the distributed modules of our GCS, b) MiddleWare Adapters that decode the communications protocols of the data received from the agents, and c) Event Generator that gathers the received decoded data and encapsulates them in the event object to deliver it to the appropriate Sink.

Sinks are the event processors/observers responsible of supporting the flexibility, re-usability and extensibility of our GCS. To achieve it, the developers only need to implement the "update" method of the "observer pattern" to ensure the compatibility of any new module with the previously deployed system in despite of its hardware or operative system. The main types of sinks are: a) Data Control that manages data reception, filtering and storage; b) Mission Control that processes and reacts to events related with the Sensors and with the tasks to be fulfilled by the UVs (e.g. showing an alarm to the Operator and reacting to it with the appropriate actions); c) Runtime Configuration Manager that deploys the selected changes in the infrastructure when the system considers that a change is required; and d) Self-Adaptive GUI that is the responsible of showing all relevant data to the operator. Being this last sink a key feature of our adaptive GCS, we are going to describe its functionality extensively in Section 4.

4 SELF-ADAPTIVE GUI

The GUI is a very important module for any application, as it is responsible of showing to the operator the data or results generated by the system. When it comes to GCSs for monitoring and controlling multiple unmanned vehicles, a well-designed GUI is the key difference for making the GCS effective or useless.

The GUI of a GCS must show huge sets of information (e.g. UVs telemetry, alarms, environment data and mission objectives) and allow the operator to take control of the vehicles at different command levels (e.g. testing UVs actuators, performing basic manoeuvres or engaging UVs in a task of the mission). In complex missions with complex UVs, several operators are often necessary to supervise the tasks related to UVs management and monitoring. If the mission involves a team of UVs, the data are multiplied for each vehicle because the operators need to monitor, at least, the relevant data for each of them. As the amount of data to observe increases, the operators need to focus on more graphical elements and get involved in more tasks, what can cause them high levels of mental workload and stress. And once operators get mentally exhausted, the only way to continue supervising the mission correctly is to replace them by a second team of restful operators, increasing the number of humans involved in the agents monitoring. Therefore, developing GUIs that maintain acceptable levels of operators' mental workload and stress is a key aspect and a continuous challenge for the modeling and implementation of GCSs. Furthermore, modeling ergonomic and operator-friendly GUIs capable of adapting to operator preferences and mission changes can not only increase operators' performance but also minimize the number of operators involved in complex missions.

In the following we present the modeling and implementation process of the ART-GCS GUI, a self-adaptive modular software that reconfigures its view at runtime in order to reduce operator's mental workload. This is possible thanks to the implementation of adaptation rules that change the position of the graphical elements for ergonomic purposes (Oreizy et al. 1999, Garlan et al. 2004) and of transparency policies that hide those elements considered less relevant for the present state of the mission (Chen et al. 2014, Mercado et al. 2016).

The Self-Adaptive GUI consists of the four modules presented in Figure 1b: the Adaptive Runtime Engine (ARE), the Controller, the View and the Configuration Database. Their main relationships can be estab-

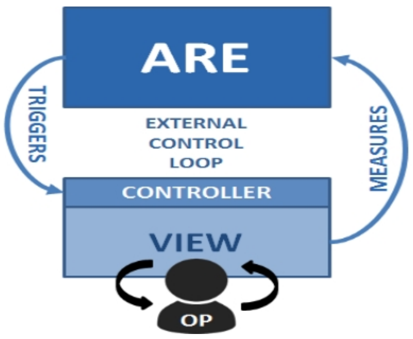


Figure 2: External Control Loop.

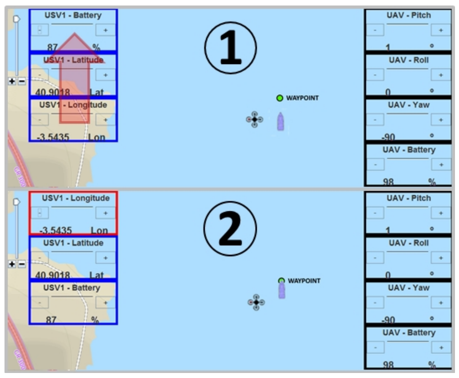


Figure 3: Red element placed in priority position.

lished through the external control loop represented in Figure 2, which states that ARE gathers information (measurements) from the View, from the operator (interacting with the View) and from the mission. Then it takes decisions based on that information and sends triggers to the Controller that modifies the View. The changes in the View affect directly operator's workload/stress and indirectly the development of the mission and therefore, future decisions of ARE. The Configuration Database is used by the different modules to reorganize the elements in the View according to each operator preferences and the global mission parameters. Further details of the functionality of these modules are presented below.

4.1 Adaptive Runtime Engine (ARE)

The Adaptive Runtime Engine (ARE) is the module that decides when a View change is needed and notifies it to the Controller to make it occur. It is a crucial part of the external control loop, as it is in charge of gathering information from the system and of analyzing it to determine the View modifications required to reduce operators workload. More specifically, 1) it monitors data related with the operators behavior (e.g their reaction time), the mission (e.g. the number of engaged Agents and the most relevant data for the task in progress) and the graphical elements on the View (e.g. their location and their number) and 2) it processes them to trigger the appropriate mechanisms on the GUI Controller.

ARE decisions, depending on the collected data and on their effects on the operators performance, can be programmed ad-hoc, or generated by Artificial Intelligence (AI) engines or Machine Learning (ML) techniques. Taking also advantage of the decoupled nature of ART-GCS design, ARE is implemented as an independent module that allows to easily change its behavior, its AI engine or its ML algorithm. This feature makes ART-GCS an ideal experimental platform for testing different approaches to help the operator to handle the big amount of data provided by a team of heterogeneous physical and/or simulated UVs in complex missions.

For the field experiment reported in this paper, the following mechanisms are implemented in ARE: 1) a *mental workload estimator* that evaluates operators workload taking into account their reaction time, their subjective mental workload and the task elapsed time; 2) a *transparency thresholder* that calculates its value based on the previously estimated operator mental workload and the number of the displayed graphic elements; and 3) a *decision tree* that assigns displaying-benefits and moving-costs to each graphic element of the GUI based on the mission task and operator's preferences.

On one hand, a change in the mental workload estimation indicates that the operators performance is either decreasing (e.g. their reaction time is rising) or increasing (e.g. they perform all their tasks in the expected time for the task). In the first case, ARE reduces the transparency threshold if possible (as it is related with the number of elements to display in the GUI), while in the second case ARE can let the transparency threshold unchanged or increment its value. In case of change, ARE notifies the new transparency threshold to the Controller, triggering the corresponding policy to adapt the View accordingly.

On the other one, the decision tree is implemented as a structure whose root-node is the initial state of the GUI, whose decision nodes represent the succession of events received and whose end nodes store the benefit/cost ratio of a graphic element after that succession of events. Besides, the decision tree changes the values of the displaying-benefits and moving-costs of each graphic element at runtime, it analyzes the events coming from the UVs or the operator, and it takes into account the requirements of the current step of the mission. For instance, when a UV reaches a waypoint, the vehicle notifies it to the GCS, and consequently to the decision tree which will decide that the operator may need to check the UV location. Hence, the decision tree increases the displaying-benefits of latitude and longitude coordinates graphical items of the corresponding UV, and notifies the change to the Controller triggering again the corresponding policy to adapt the View accordingly (see example in Figure 3).

4.2 GUI Controller

The Controller is the executive module that performs on the View the changes determined by ARE. It is where the policies related with the changes on the View are implemented. Moreover, it updates the Priority Vectors (displayed in Figure 1b) where the graphical elements associated to each UV in the mission are sorted according to each element priority, which is calculated as the ratio of its displaying-benefit to its moving-cost.

More specifically, when ARE notifies a change in the transparency threshold, the transparency policy is triggered and the Controller refreshes the View. To do it, it hides those elements of each Priority Vector that fall below the transparency threshold and shows the ones that appear above. In other words, the transparency policy decreases/increases the number of elements displayed in the GUI according to the transparency threshold that ARE calculates taking into account the increment/reduction of the operators mental workload.

When ARE notifies a change in the displaying-benefit or moving-cost of any element of the View, the GUI adaptation process is triggered, the Priority Vector re-sorted and the View refreshed by the Controller. This process relocates the graphic elements within the View margins according to their priority order. In other words, the GUI adaptation policy accommodates the position and size of the elements of the View at runtime (as the example of Figure 3 shows) according to the needs of the task in progress.

Finally, the adaptation policy of the Controller also takes into account the preferences of the operator (e.g. best position to observe, colors) to display the elements on the View. In other words, graphic elements associated to equal Priority Vectors with the same priority threshold can be represented differently in the View to be able to adapt not only to the current state of the mission but also to the stress and ergonomic preferences of the operator.

4.3 Adaptive View

The View is the piece of software that displays the information to operators and lets them interact with the rest of the system (commanding UVs at individual or mission level, requesting data, etc.). It consists of a group of UVs and mission commanders, a set of graphic elements considered relevant to carry out the mission, and a map where the current location of the UVs and some information related with the mission (e.g. the waypoints or UAvs trajectories) are drawn.

As we have already seen, the number of the elements displayed on the View, as well as their location, size and color, are changed by the Controller according to the decisions taken by ARE to accommodate the operator mental workload and the information-displaying requirements of the mission. The final purpose of these changes is to help operators to focus comfortably on the relevant information at each stage of the mission while maintaining their awareness. Finally, it is worth noting that the View can also adapt to direct operator requests, related with which information to show and where. These requests will override, if necessary, both ARE and Controller display decisions. Besides, they will be stored in the GUI Configuration Database to analyze them afterwards and determine if they could be useful in future missions.

4.4 GUI and Operators Configuration Database

This is a persistent storage module that keeps the main configuration parameters of the GUI and the data of each operator's profile and preferences. In a nutshell, it provides information on how to display graphic elements in the most ergonomic possible way for a specific operator running a specific task/mission.

On one hand, the main configuration parameters are basically a guide of how to generally position graphic elements on the View. It stores the limits of the screen, the areas to locate graphic elements, and the default higher priority positions (e.g. up-left). It also stores the missions decision trees (which incorporate also the displaying-benefits and moving-costs of each graphic element) and the expected elapsed time of each task of the mission.

On the other one, when a new operator is registered in ART-GCS, a profile is created for him/her, storing his/her username and password for security, as well as other data related to ergonomic preferences like the chosen priority positions, colors, etc. During the development of the missions, the database also stores information about the operator behavior to analyze it afterwards and adapt ARE decisions (i.e. modify its decision tree on the database) to the operator requests and preferences in specific tasks/missions. For example, if the decision tree usually prioritizes the latitude and the longitude of the UVs arriving at waypoints and one operator often requires also to observe the UV orientation, at some point the decision tree of these operator will be modified to make the ARE prioritize the UV orientation for the same operator at the same situations.

5 USE CASE

The capability of ART-GCS to manage multi-UV missions has already been shown in (Bonache-Seco et al. 2018), where the focus of the experiment was centered on describing, from the software architecture point of view, the runtime interaction of two UVs in a potentially dangerous leader-follower maneuver. On that experiment, the leader (simulated UV to reduce the crashing risks) tries to pursue a lemniscate curve (de la Cruz et al. 2015), while the follower (physical UV to see if the manoeuvre is feasible) tries to remain at a given distance and angle with respect the moving leader. In this section we describe a new experiment from the perspective of the GUI adaptability, drawing the readers attention to the effects in the View when ARE triggers the adaptation and transparency policies of the Controller at different steps of the mission.

The new mission consists on deploying a team of two Unmanned Surface Vehicles (USV1 and USV2) and one Unmanned Aerial Vehicle (UAV) for searching and containing an oil-spill on a water surface. The UAV performs the oil-spill search task, while the USVs carry out a cooperating maneuver to enclose the spill with a containment boom. To conduct the experiment in the field, we deploy two physical USVs on the water, and command them to reach and stop at a defined waypoint, while dragging a boom. Once both USVs are stopped, the UAV has to take off from USV1, find the oil spill while performing a lawnmower searching maneuver, and return to USV1 position to land. As taking off and landing on a mobile platform over the water surface is specially risky for the UAV, we prefer to simulate its behavior in its initial integration phases

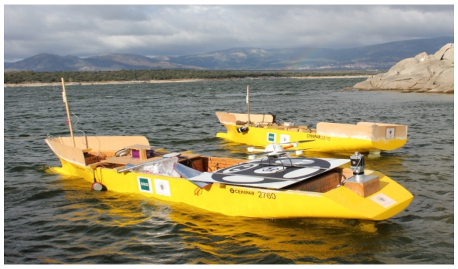


Figure 4: Experimental UVs.

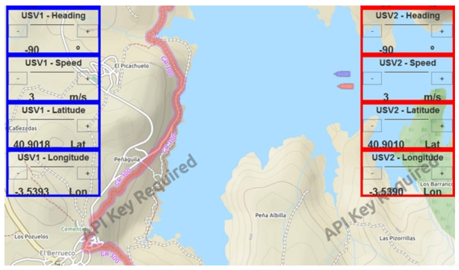


Figure 5: Initial View of the GUI.

in the real mission. Besides, simulating the UAV sensing behavior, we can make it also detect unreal oil-spills during unpolluting experimental tests. Finally, and after simulating UAV landing, the USVs have to perform a maneuver for keeping the oil-spill enclosed with a boom (until it is skimmed from the water).

More specifically, the physical USVs of the mission are the two scale boats of 4 and 5 meters presented in Figure 4, which have been built by the Canal de Experiencias Hidrodinámicas de El Pardo, Spain (CEHIPAR) and which are controlled with a Beckoff C6920 built-in computer with TwinCAT, a real time system with embedded Windows Operating Systems. Moreover, USV1 is equipped with a VectorNav VN-200 Inertial Measurement Unit (IMU) with Global Positioning System (GPS), and USV2 with a CodaOctopus F180 IMU with GPS. The simulated UAV models a Microdrone MD4-200 (our physical UAV, also displayed in Figure 4) over a computer with RT-Linux. The communications between the UVs and the GCS, this last deployed in a laptop (Intel Core i7-4500U 2x1.8GHz and 4Gb RAM) with Windows 8.1 OS, are performed through a Wi-Fi wireless network (of 1.5 to 2 Km of range) formed by 3 PicoStation M (connected each to the computer of each UVs) and a Ubiquiti NanoStation M2 antenna (connected to the GCS).

The objectives of this experiment are: 1) verifying that the physical USVs can perform correctly their different maneuvers, 2) checking the communications and the correct exchange of information, through the Channel Manager, between the physical and simulated UVs, 3) testing if one operator is capable of supervising and managing all the tasks of the mission, and 4) analyzing the feasibility of the self-adaptive GUI and the implemented adaptive and transparency policies in the scope of this mission, not only during the transition from two to tree UVs but also from the point of view of the operator mental workload influence.

In the first part of the experiment, USV1 and USV2 are deployed on the water surface to begin a cooperation maneuver where USV2 acts as the leader and USV1 as the follower. During this stage of the mission, both USVs must approach a defined waypoint, maintaining an appropriate distance between them while the GUI of ART-GCS displays the information presented in Figure 5. This occurs since ARE sets the transparency threshold to 4 (making the Controller display only 4 graphic elements per vehicle, the ones of USV1 in blue in the upper-left side and the ones of USV2 in red in the upper-right one) because the estimated mental workload of the only operator supervising the mission is medium (as a consequence of having only two UVs active in a mission that is being successfully performed so far).

Once the USVs stop at the defined waypoint, the next phase of the mission starts. In this part of the experiment, USV1 and USV2 remain stopped and the simulated UAV takes off (assuming the coordinates provided by USV1). Next, the UAV starts performing (i.e. simulating) the lawnmower searching maneuver to locate the oil-spill with the purpose of sending its coordinates to the USVs for its posterior boom containment. Finally, the UAV requests USV1 position to return and perform a simulated landing over it. During this phase of the mission there are several changes in the GUI. When the UAV incorporates itself to the mission, 1) it is integrated in the adaptive infrastructure of ART-GCS, and 2) ARE, which is monitoring the mission and operator at runtime, detects its presence and adds its graphic elements (position on the map and relevant telemetry data) on the screen, as Figure 6 displays. At this point, as the threshold priority is still set to 4, the GUI displays the four more relevant pieces of information of the three UVs (centering the data of the UAV

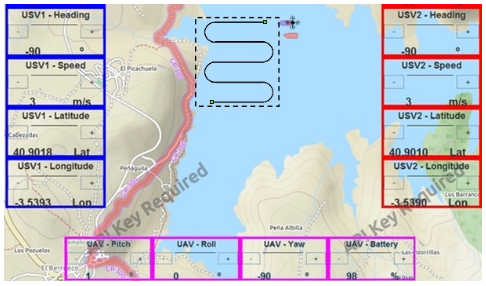

USV1 - Heading -90 °
USV1 - Speed 3 m/s
USV1 - Latitude 40.9018 Lat
USV1 - Longitude -3.5393 Lon
USV2 - Heading -90 °
USV2 - Speed 3 m/s
USV2 - Latitude 40.9010 Lat
USV2 - Longitude -3.5390 Lon
UAV - Pitch 1 °
UAV - Roll 0 °
UAV - Yaw -90 °
UAV - Battery 98 %

Figure 6: GUI View after UAV takes off.

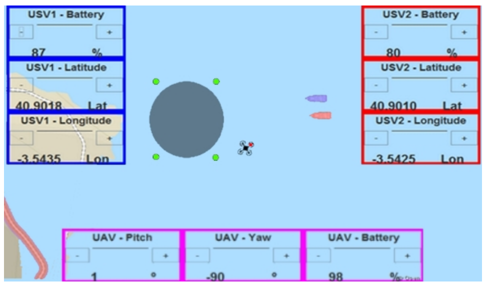

USV1 - Battery 87 %
USV1 - Latitude 40.9018 Lat
USV1 - Longitude -3.5435 Lon
USV2 - Battery 80 %
USV2 - Latitude 40.9010 Lat
USV2 - Longitude -3.5425 Lon
UAV - Pitch 1 °
UAV - Yaw -90 °
UAV - Battery 98 %

Figure 7: GUI View while UAV returns to land.

in the lower part of the screen). The big amount of data displayed in the View and the stress related with the UAV take off rises the reaction time of the operator. This makes ARE increment its estimation of the operator mental workload and consequently reduce in a unity the transparency threshold of the GUI. This last change in the transparency threshold value triggers the Controller's transparency mechanism, which reduces the information displayed for each UV from the 4 elements (the previous threshold value) of Figure 6 to the 3 ones (the current threshold value) of Figure 7. Thanks to these changes, the operator has less elements to focus on, and the measurements of his reaction time and estimated mental workload decrease, allowing to continue the mission under his/her unique supervision. Additionally, the Controller is also triggered with information related with new displaying benefits of the graphical elements of each UV, re-sorts the information of its Priority Vectors, and re-locates the most relevant information in the best positions inside the GUI (upper positions for the USVs and left-bottom positions for the UAV).

After the UAV performs its simulated landing maneuver, the GCS planner calculates a waypoints route for the USVs and asks them to perform the oil-spill enclosing maneuver. The waypoints of this route are selected from the ones gathered during the simulated UAV oil-spill location maneuver. Moreover, after this point, the mission returns to its initial configuration, with only two USVs being monitored by a more relaxed operator because the oil-spill enclosing maneuver has been successfully tested in previous experiments. Hence, the transparency threshold is set back to 4 by ARE, which triggers the adaptation and transparency mechanism of the Controller to re-configure the View with the graphical elements of the four more relevant pieces of information of each USV during the current phase of the mission. Hence, the type and organization of the displayed telemetry data at this stage is equivalent to the one already shown in Figure 5.

6 CONCLUSION

ART-GCS, the Adaptive Real-Time Ground Control Station presented in this paper, has been modeled, designed and implemented to help operators to handle complex missions performed by heterogeneous sets of physical and/or simulated UVs, while avoiding high levels of mental workload and stress. Its adaptation capabilities allow ART-GCS to handle at runtime changing sets of UVs and to "intelligently" accommodate the information displayed in its GUI to the mission phase and to the operators performance. The field experiment presented in the paper, involving different sets of heterogeneous real and simulated agents, shows the suitability and adaptability of ART-GCS to a real world complex mission.

In the future, we have to continue performing field experiments to demonstrate the flexibility and adaptability of ART-GCS to the reaction time and mental workload of different operators, and to determine the correct ranges of the transparency threshold and their effects in the operator performance (since it can be equally stressing to observe too much or not enough data). Moreover, and taking advantage of the decoupled nature of ART-GCS, we will modify the mechanisms that trigger the changes of the information displayed to the operators, by incorporating and testing different Artificial Intelligence or Machine Learning techniques to the Adaptive Runtime Engine of its self-adaptive GUI. Finally, the data gathered within the scope of the experiments in progress will let us 1) compare operator's performance (reaction time, mental workload, etc.) using non-adaptive and adaptive GUIs (with different IA/ML algorithms ARE), 2) provide qualitative and quantitative evaluations of the different approaches to validate them formally, and 3) improve ART-GCS, and consequently the performance of its operators.

ACKNOWLEDGMENTS

This work is supported by the Spanish Ministry of Economy and Competitiveness (MINECO) through the funding of DPI2013-46665-C2-1-R project.

REFERENCES

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

Bonache-Seco, J., J. Lopez-Orozco, E. Besada-Portas, and J. L. Risco-Martín. 2018. "Adaptive Event Driven Framework for Real Time Multi-Agent Missions". *22nd IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2018)*.

Bürkle, A., F. Segor, M. Kollmann, and R. Schönbein. 2011. "Universal Ground Control Station for Heterogeneous Sensors". *Journal On Advances in Telecommunications, IARIA* vol. 3 (3), pp. 152–161.

Chen, J., K. Procci, M. Boyce, J. Wright, A. Garcia, and M. Barnes. 2014. "Situation awareness-based agent transparency(No. ARL-TR-6905)".

Cheng, S.-W., D. Garlan, and B. Schmerl. 2006, May. "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.

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

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

Etzion, O. 2005. "Towards an Event-Driven Architecture: An Infrastructure for Event Processing Position Paper". *Lecture Notes in Computer Science book series (LNCS, volume 3791)*.

Filipponi, L., A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci. 2010. "Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors". *Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM)*.

Garlan, D., S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. 2004, October. "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure". *Computer (IEEE Computer Society)* vol. 7 (Issue: 10), pp. 48–54.

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

Heo, J., S. Kim, and Y. Kwon. 2016. "Design of Ground Control Station for Operation of Multiple Combat Entities". *Journal of Computer and Communications* vol. 4, pp. 66–71.

Hong, W. E., J. S. Lee, L. Rai, and S. J. Kang. 2005. "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)*.

Jovanovic, M., and D. Starcevic. 2010. "Software Architecture for Ground Control Station for Unmanned Aerial Vehicle". *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*.

Jovanovic, M., D. Starcevic, and Z. Jovanovic. 2008. "Improving Design of Ground Control Station for Unmanned Aerial Vehicle: Borrowing from Design Patterns". *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*.

Laddaga, R., and P. Robertson. 2004. "Self adaptive software: A position paper". In *SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, Volume 31, pp. 19. Citeseer.

Lindemuth, M., R. Murphy, E. Steimle, W. Armitage, K. Dreger, T. Elliot, M. Hall, D. Kalyadin, J. Kramer, M. Palankar, K. Pratt, and C. Griffin. 2011, June. "Sea Robot-Assisted Inspection". *IEEE Robotics Automation Magazine* vol. 18 (2), pp. 96–107.

Mercado, J. E., M. A. Rupp, J. Y. C. Chen, M. J. Barnes, D. Barber, and K. Procci. 2016. "Intelligent Agent Transparency in Human-Agent Teaming for Multi-UxV Management". *Human Factors* vol. 58 (3), pp. 401–415.

Michelson, B. M. 2006. "Event-Driven Architecture Overview". *Patricia Seybold Group Research Service (2006)*.

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

Oreizy, P., M. M. Gorlick, and R. N. Taylor. 1999. "An architecture-based approach to self-adaptive software". *Intelligent Systems and their applications* vol. 14 (3).

Patterson, M. C., A. Mulligan, and F. Boiteux. 2013. "Safety and security applications for micro-unmanned surface vessels". In *2013 OCEANS-San Diego*, pp. 1–6. IEEE.

Ribas, D., N. Palomeras, P. Ridao, M. Carreras, and A. Mallios. 2012. "Girona 500 auv: From survey to intervention". *IEEE ASME Transactions on Mechatronics* vol. 17 (1), pp. 46–53.

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

AUTHOR BIOGRAPHIES

JUAN A. BONACHE-SECO is a Ph.D. Student in the University Complutense of Madrid. His research interests include Self-Adaptive Software, Architectural Frameworks, GCSs for Unmanned Vehicles and Robotics. His email address is jabonache@ucm.es.

JOSÉ A. LOPEZ-OROZCO is an Associate Professor in the University Complutense of Madrid. He holds a Ph.D. in Physical Sciences from the same University. His research interests include Multisensor Data Fusion, Control and Planning of Unmanned Vehicles, and Robotics. His email address is jalo@ucm.es.

EVA BESADA-PORTAS received her PhD from the University Complutense of Madrid and is currently an Associate Professor at the same university. Her research interests include uncertainty modeling, optimal control and planning of unmanned vehicles. Her email address is ebesada@ucm.es.

JOSÉ L. RISCO-MARTÍN received his Ph.D. from Complutense University of Madrid, and currently is Associate Professor in the Department of Computer Architecture and Automation at Complutense University of Madrid. His research interests include computer aided design, optimization and discrete event simulation. His email address is jlrisco@ucm.es.