JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 8, July 2011

Autopilot Abstraction and Standardization for Seamless Integration of Unmanned Aircraft System Applications

Pablo Royo[*] , Enric Pastor[†] , Cristina Barrado[‡] , Eduard Santamaria[§] , Juan Lopez[¶] , Xavier Prats[#] , and Juan Manuel Lema[**]
Castelldefels School of Technology (EPSC) 08860, Castelldefels, Spain

DOI: 10.2514/1.52672

Nowadays many autopilot manufacturers are available in the commercial market for fixed wing small/mini Unmanned Aircraft System. Several autopilot configurations exist with a wide variety of selected sensors, sizes, control algorithms, and operational capabilities. However, selecting the right autopilot to be integrated in a given Unmanned Aircraft System is a complex task because none of them are mutually compatible. Moving from one autopilot to another may imply redesigning from scratch all the remaining avionics in the Unmanned Aircraft System. This paper presents the Virtual Autopilot System to facilitate exploitation of data obtained from the autopilot to be used by other applications on board. At the same time, it provides a hardware-independent interface that isolates payload and mission components from the autopilot specificities, thus eliminating dependencies on a particular autopilot solution. This subsystem is integrated into an Unmanned Aircraft System mission-oriented architecture called Unmanned Aircraft System Service Abstraction Layer, which promotes the development of automated concepts of operation keeping the Unmanned Aircraft System pilot fully under control. The VAS and its surrounding architecture have been implemented for a variety of autopilots, ranging from the commercial AP04 from *UAV NAVIGATION*, to the Paparazzi autopilot and even autopilots for ground-based vehicles. In all cases the selected Virtual Autopilot System interface was maintained, overall capabilities increased due to the flight-plan and mission-oriented perspective offered by the surrounding architecture, and development times exponentially reduced as the Virtual Autopilot System design is consolidated. This wealth of experimentation demonstrates that employing a standardized interface

Received 6 October 2010; accepted for publication 16 March 2011.

[*] Lecturer, Castelldefels School of Technology (EPSC), Office C4-010, UPC. Av. Esteve Terradas, 7. 08860, Castelldefels, Catalonia (Spain). AIAA Member, proyo@ac.upc.edu

[†] Professor, Castelldefels School of Technology (EPSC), Office C4-002, UPC. Av. Esteve Terradas, 7. 08860, Castelldefels, Catalonia (Spain). AIAA Member.

[‡] Professor, Castelldefels School of Technology (EPSC), Office C4-013, UPC. Av. Esteve Terradas, 7. 08860, Castelldefels, Catalonia (Spain).

[§] Assistant professor, Castelldefels School of Technology (EPSC), Office C4-005, UPC. Av. Esteve Terradas, 7. 08860, Castelldefels, Catalonia (Spain).

[¶] Professor, Castelldefels School of Technology (EPSC), Office C4-010, UPC. Av. Esteve Terradas, 7. 08860, Castelldefels, Catalonia (Spain).

[#] Assistant Professor, Castelldefels School of Technology (EPSC), Office C3-104, UPC. Av. Esteve Terradas, 5. 08860, Castelldefels, Catalonia (Spain).

[**] Researcher, Castelldefels School of Technology (EPSC), Office C4-015, UPC. Av. Esteve Terradas, 7. 08860, Castelldefels, Catalonia (Spain).

facilitates the integration of new Auto Pilots, but at the same time capabilities are not only maintained but dramatically increased by developing flight-management systems on top of the Virtual Autopilot System standard interface.

I. Introduction

CURRENTLY, Unmanned Aircraft Systems (UASs) are mostly being used for military applications such as in [1], but with the evolution of avionics technology, a huge market in civil applications is now emerging [2–5]. However, there is a lack of hardware and software support to effectively develop these potentialities in the civil domain. No commercial solution exists today that provides support for all these applications.

Economic efficiency requires the same UAS to be able to operate in different application domains. Therefore, one of the current challenges in UAS research is to define a hardware/software UAS framework that is sufficiently flexible and reusable to be operated in a wide range of civil applications. To be competitive in the civil market, this framework must provide fast prototyping and be affordable.

In previous work, a flexible, reusable, and distributed hardware/software UAS architecture was introduced in order to support the development of different UAS civil missions [6,7]. This architecture works as an abstraction layer, called UAS Service Abstraction Layer (USAL), that allows the easy and fast design of missions and enables the usability of the system in a cost-effective way.

During the development of the USAL some relevant questions arose. Can dependencies on a particular autopilot solution be eliminated? Is it possible to provide a hardware-independent interface that isolates mission and payload components from the autopilot specificities? From the study of several commercial UAS autopilots, we have identified three clear drawbacks that limit the effective integration with the mission and payload control inside the UAS:

1) Some sort of standardization and adaptation is needed, focused on the UAS mission instead of the UAS flight.
2) Exploitation of autopilot telemetry by other applications in the UAS is complex and depends on the autopilot.
3) The flight plan definition available is just a collection of waypoints that are statically defined or handmanipulated by the UAS operator.

This paper presents a key component of the USAL architecture called the *Virtual Autopilot System (VAS)*. The VAS is a component designed to facilitate exploitation of data obtained from the autopilot, offering flight-related data flows to other systems on board the UAS. The VAS provides a hardware-independent interface that isolates mission components from the autopilot specificities. On one side the VAS interacts with the selected autopilot and therefore needs to be adapted to its peculiarities. On the other side, it interacts with most USAL components and therefore needs to offer a well-defined standardized interface and capabilities. The VAS operates similarly as drivers work on operating systems (OSs), removing the implementation details from actual autopilot users.

The VAS also benefits from a great effort to properly define common autopilot states and from a well-defined concept of operation designed to increase the level of automation and simplify the pilot interaction. At the same time, the VAS retains simple navigation capabilities for traditional operations. The VAS and USAL go beyond the development of specific navigation and/or mission solutions like the ones introduced in [1,8–13], by proposing a common framework on top of which particular applications can be further developed. Embedded safety modes are also proposed to facilitate the integration of the UAS in the airspace. Finally, the Human Machine Interface (HMI) aspects are considered to be of the utmost relevance, and therefore the VAS is designed to have a powerful ground control interface.

The paper is organized as follows. Section II describes existing UAS autopilots and highlights their disparity in capabilities and interfaces, thus motivating the development of this work. Section III outlines the USAL architecture and the communication middleware that supports its distributed paradigm. In Sec. IV, an overview of the VAS is provided and its architecture introduced. In Sec. V, VAS operational mode design and its interface with the rest of the USAL services are described. Some implementation details are provided in Sec. VI. Section VII details simulation-based experimental results and field tests on real prototypes. Finally, our conclusions and planned future work are presented in Sec. VIII.

II. Previous Work

The heart of a UAS is the autopilot. Without it, the UAS cannot fly autonomously. The autopilot implements the control layer and applies control laws on the different UAS actuators to direct its flight. It is essential to understand how autopilots work, what kind of inputs they support, and what are their capabilities in order to design a suitable UAS open architecture on top of them. There is a considerable amount of autopilots designed for UASs such as Piccolo autopilot [14], AP04 autopilot [15], MicroPilot UAV autopilot [16], Paparazzi autopilot [17], ArduPilot [18], and Aalborg University autopilot [19] or [20,21], and the list is constantly growing. It is not the purpose of this section to provide an exhaustive survey of UAS autopilots. The following paragraphs describe a few of the available autopilots and provides a short description of their technical characteristics in order to identify their drawbacks and limitations.

The most relevant studied autopilots have been Piccolo from Cloud Cap Technology, AP04 from *UAV-NAVIGATION*, and Paparazzi from the ENAC. Piccolo and AP04 are examples of commercially available products, whereas Paparazzi is a research autopilot. Piccolo and AP04 support autonomous operations such as launch, navigation, or land, but no generalized concept of operation to perform those maneuvers. All of them also provide a mechanism for storage of telemetry data in real time. Piccolo provides big binary streams of telemetry data, which are hard to process by other applications on board. Both AP04 and Paparazzi organize telemetry in data packets sent at different rates, which are also too big to be easily exploited. All three autopilots offer input and output capabilities in order to interact with the payload. However, the payload components do not have any mechanism to interact with the flight plan and coordinate both activities. This approach is very restrictive in terms of flexibility and potential for automation. Almost every autopilot has all the standard set of sensors (Inertial Measurement Unit (IMU), magnetometer, air data system, Global Positioning System (GPS) receiver) and in some cases, for example with AP04, also featuring dual CPU redundancy. Paparazzi is a combination of infrared thermopile and inertial measurement for attitude sensing, providing a robust and accurate attitude estimate that requires no ground calibration. With regard to the flight plan capabilities, they only provide a collection of waypoints hand-manipulated by the pilot in command (PiC).

During this study, three drawbacks were clearly identified that limited the autopilot's effective integration with the mission and payload control inside the UAS:

1) Depending on the implementation and capabilities of each autopilot, the autopilot's behaviors or states will differ. Some sort of standardization and adaptation is needed, focused on the UAS mission instead of the UAS flight.
2) Exploitation of autopilot telemetry by other applications in the UAS is complex and depends on the autopilot. Autopilot telemetry is typically designed just to monitor the state and position of the UAS, and is not to be used by third party applications.
3) The flight plan definition available in most autopilots is just a collection of waypoints that are statically defined or hand-manipulated by the UAS operator. This approach severely limits the flexibility of the system because no possible interaction exists between the flight plan, the mission itself and the payload operated by the UAS.

Further details on this analysis cannot be publicly reproduced due to the limitations imposed by the *nondisclosure agreements* with the proprietary companies. It is worth noting that a wide range of academic autopilots are under development, but most of them are mainly focused on the research of control theory and algorithms rather than on the UAS operation or the interface with the remainder of systems within the UAS architecture.

III. USAL System Overview

The USAL has been introduced as a flexible, reusable, and distributed hardware/software architecture to support the development of different UAS civil missions [6,7]. This architecture works as an abstraction layer that simplifies the design of UAS missions and enables the reusability of the system in a cost-effective way. The existence of an avionics framework package specifically designed for UASs may alleviate the development effort and cost, reducing them to a simple configuration or parameterization.

The design of this avionic framework started with the definition of its functional requirements. These requirements were defined by the type of UAS (basically mini or tactical UAS) and by the mission objectives. From the study and definition of several UAS missions, we identified the most common functionalities that are present among them.

Elements such as autopilots, cameras, mission control engines, payload controllers, contingency managers (CMs), and data storage are needed in almost all UAS missions. Therefore, it is a natural approach to provide a framework for defining and standardizing all these components, capturing common requirements and functionalities.

The goal of USAL is to reduce development effort when creating a new UAS system, providing standardization but also flexibility to the development of all systems required to implement the UAS mission. The USAL is the set of available *services*, running on top of a middleware called Middleware Architecture for Remote Embedded Applications (MAREA) [22], that provide support to most types of UAS missions. The USAL also defines service interrelations as the basic starting point for further development by users. Functionalities like enhanced flight plans, a mission control engine, data storage, autopilot management, and contingency management are offered.

The USAL can be compared to an OS. Computers have hardware devices used for input/output operations. Every device has its own particularities and the OS offers an abstraction layer to access such devices in a uniform way. Basically, it publishes an Application Program Interface, which provides end-users with efficient and secure access to all hardware elements.

The USAL considers sensors and in general all payload as hardware devices of a computer. It is a software abstraction layer that gives facilities to end-users' programs to access the UAS payload. It also provides many other useful features designed to simplify the complexity of developing the UAS application. It defines a collection of basic services that comprises a minimum common set of elements that are needed in any UAS. A number of additional services have been identified as "highly useful" in most UAS application. The USAL intends to provide a framework in which the UAS developer can easily exploit these services, but at the same time provide guiding directives on how new ad hoc services should be created.

A. USAL Service Categories

The USAL is composed of a set of distributed services running on top of a middleware that have been organized and divided into several categories. Services are grouped into the same category when they cooperate toward the same objective, such as flight, mission, payload, or awareness. However, not all of them have to be present in every UAS or in every mission. Only those services required for a given configuration/mission should be present and/or activated in the UAS.

Available services have been classified into four categories (see Figs. 1 and 2):

1) Flight Services: Services responsible for basic UAS flight operations: autopilot, flight plan management, basic monitoring, contingency management, etc.

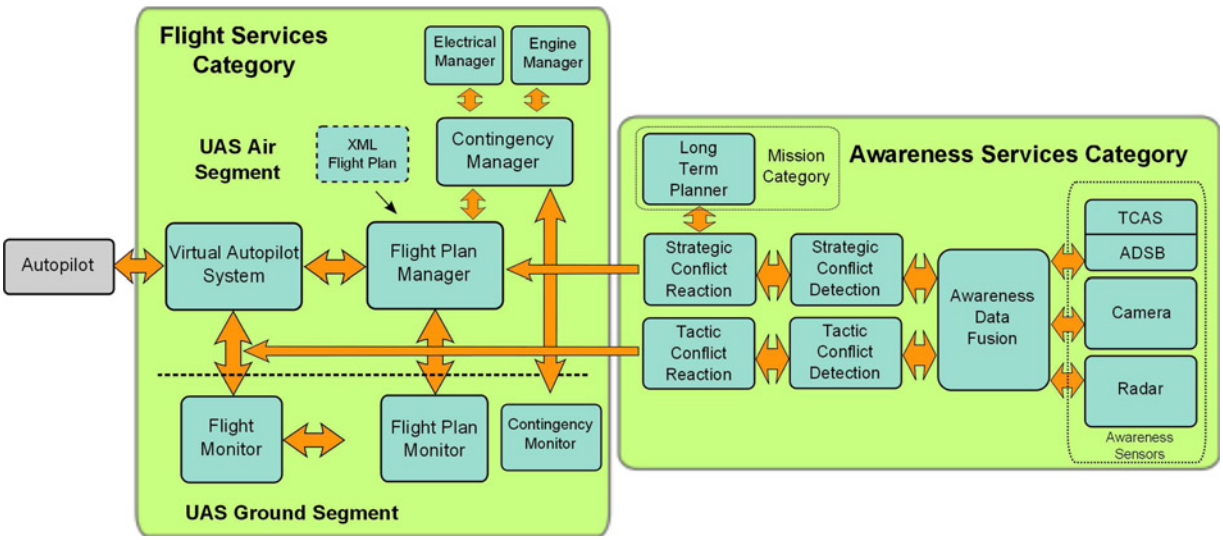

Fig. 1 Overview of the USAL flight and awareness category interaction.

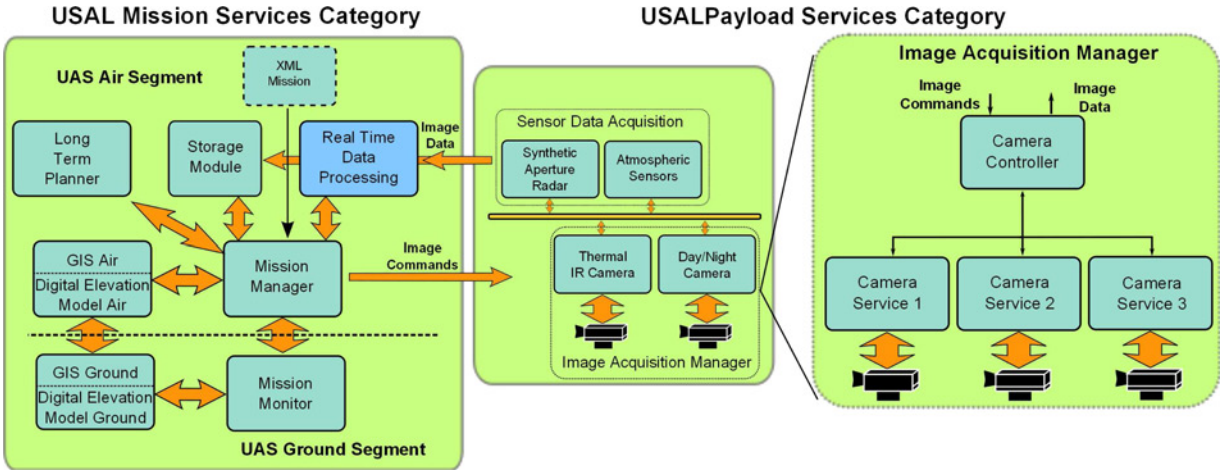

Fig. 2 Overview of the mission and payload category interaction.

2) Mission Services: Services responsible for carrying out the actual UAS mission.
3) Payload Services: Specialized services interfacing with the input/output capabilities provided by the payload on board the UAS.
4) Awareness Services: Services responsible for the safe operation of the UAS with regard to its integration with other airspace users or terrain avoidance.

B. USAL Flight Category

The flight category is responsible for basic UAS flight actions. Each one of the services in this category has a specific goal related to UAS airworthiness. Flight services target the following objectives:

1) To abstract autopilot details and peculiarities to the rest of the system.
2) To extract internal sensor information from the autopilot and offer it to other services for its exploitation during the UAS mission.
3) To provide a common flight plan definition, significantly improving current commercial autopilot capabilities.
4) To provide status monitoring capabilities and automatic contingency management for an efficient emergency response (including the monitoring of the electrical and engine subsystems).

Figure 1 depicts the fundamental components in the flight services category as well as the major relationships among them.

The main services in this category are the VAS and the *Flight Plan Manager* (FPMa). These components interact with the ground segment through the *Flight Monitor* (FM) and the *Flight Plan Monitor* (FPMo) services. Any electrical or mechanical contingency is managed by the CM service. This service can interact with the UAS directly using the VAS or through the FPMa.

The VAS is a service that interacts with the selected autopilot and therefore needs to be adapted to its peculiarities. The VAS is a service provider that offers a number of information flows to be exploited by the USAL.

The flight planning capabilities of all autopilots are dissimilar, but they are generally limited to simple waypoint navigation. From the point of view of the current missions, using a simple waypoint-based flight plan may be extremely restrictive. Alongside the VAS, we have developed an FPMa designed to extend current autopilot flight plan capabilities [23]. The FPMa offers an almost unlimited number of waypoints, waypoint grouping, structured flight plan phases with built-in emergency alternatives, mission-oriented legs (that specify the path that the aircraft must follow in order to reach a destination waypoint) with high-level semantics like repetitions, parameterized scans, etc. Flight plans are specified using an *Area Navigation* (RNAV) inspired [24] XML formalism.

Area Navigation is an Instrument Flight Rules navigation method that allows to fly routes that can be defined between arbitrary waypoints within the coverage of a network of navigation beacons or within the limits of a self-contained positioning system, or a combination of both.

The *Engine and Fuel Manager System* and the *Electrical Manager* are in charge of monitoring the engine and electrical parameters of the UAS in order to detect and notify potential contingencies. Besides monitoring functions, both services can also estimate the time remaining to carry out the mission under nominal conditions.

Employing the USAL permits addressing the safety issues related to UAS operation from a high-level perspective. The combination of a dedicated service with preplanned contingency reactions embedded both within the flight plan and the VAS provides a powerful safety mechanism. The CM is responsible for collecting status information related to multiple sources, such as engine, electrical, fuel, etc., identifying that some type of contingency has evolved, and deciding which type of reaction is required. The reaction will be more drastic depending on the type of contingency [25]: in some situations the CM will reconfigure parts of the USAL to continue the UAS mission under degraded conditions; whereas in others the FPM and VAS will be commanded to execute some emergency operation, e.g., an immediate return to base, or selecting a predefined alternative landing site or executing an emergency flight termination procedure.

Note, however, that the autopilot and VAS need to retain a certain level of contingency capabilities. At each step of the flight, a preplanned basic reaction maneuver will be sent from the FPM to VAS and from there to the Auto Pilot (AP) itself. In case the USAL communication infrastructure fails, VAS will interpret this as a lost-link situation and react following the preplanned termination procedures (in that case the reaction scheme will be much simpler than that available through the overall USAL). If VAS itself fails, the AP will identify the lost link and execute the preplanned termination. Optionally, the AP may use a dedicated link with ground control to override the overall VAS interface. This mechanism is not hard to implement and provides yet another safety net, although specific for each AP.

A certain level of control and monitoring is required from the ground segment so that the PiC is able to supervise the UAS operation at all times. However, different operator profiles should exist depending on the functionalities that require monitoring. In the USAL framework, we have identified three different operator profiles: pilot, flight plan controller, and mission controller. The flight service category includes support for both direct flight control through the FM service, and flight plan modification through the FPMo service. In addition, special attention is paid to contingency situations during the mission. Therefore, additional monitoring is available for both types of operation (manual and flight plan based) in order to manage in-flight contingencies through the *Contingency Monitor*. This service may be embedded into the FM.

The pilot interface aspects of the USAL are also developed, but will not be described in detail as they are out of the scope of this paper. Figure 3 shows a portion of the FM HMI. This pilot-oriented interface will offer traditional information about telemetry, real-time video, basic flight plan tracking, and the system state. However, the FM also includes diverse interchangeable components that can be accessed through an auxiliary screen (see Fig. 4). Among others, the available components include: basic flight plan information, VAS state management, alarms, support for taxi operations, support for takeoff and landing operations [26], payload configuration, etc.

The FPMo is in charge of real-time monitoring of the entire UAS flight plan. Unmanned Aircraft System Service Abstraction Layer flight plans are defined as structured collections of both basic and sophisticated legs (including alternatives, repetitions, and other parameterized legs) [23]. The FPMo allows to monitor and control the progress of the flight plan, performing dynamic updates on the fly by modifying those parameters rather than updating individual waypoints. This dynamic flight plan formalism allows to describe highly complex behavior by exploiting the aforementioned legs and the update mechanism. However, the formalism also allows to embed emergency flight plan alternatives and takeoff/landing parameters to dynamically choose between airfields. The appropriate design of these HMI interfaces is ongoing research that will be described in future work.

C. Middleware Architecture for Remote Embedded Applications

The MAREA is a middleware system used to communicate different services over a Local Area Network (LAN). The MAREA provides an execution environment with communication channels and common functionalities. The role of each service is expressed by the action of publishing, subscribing, or both simultaneously; in this way, the publish/subscribe model eliminates complex network programming for distributed applications and makes it easy to implement an embedded service. The MAREA offers the localization of the other services and manages their discovery within the network; it handles all the transfer chores, message addressing and retransmission, data delivery, flow control, etc.

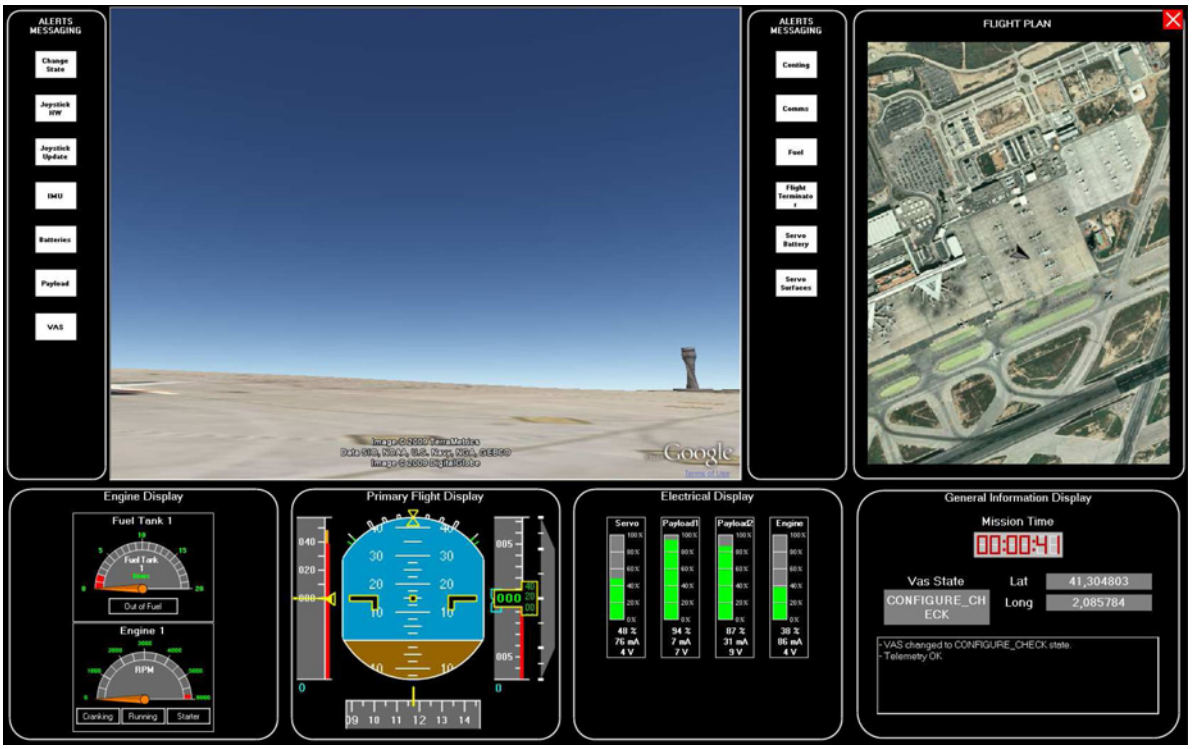

Fig. 3 Flight Monitor Service: main screen.

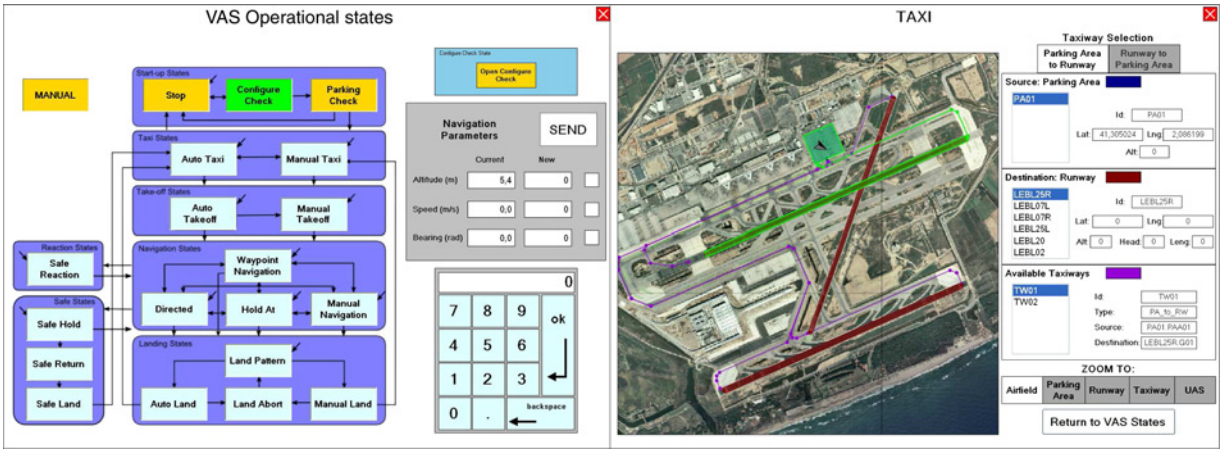

Fig. 4 Flight Monitor Service: auxiliary interfaces provided through a secondary Multi-Function Display (MFD).

The MAREA promotes the Service Oriented Architecture (SOA) publish/subscribe model for sending and receiving data. The MAREA is responsible for delivering the information to all subscribers that declare an interest in a certain flow of data. Information exchange is carried out through four communication primitives, which are called variables, events, remote invocations, and file transmissions:

1) Variables may be sent at regular intervals or each time a substantial change in their value occurs. The system should be able to tolerate the loss of one or more of these data transmissions. Example of variables are flight, engine, or electrical telemetry.

2) Like Variables, events also follow the publication/subscription paradigm. The main difference, when compared with variables, is that events guarantee that all subscribers will receive the sent information. Events are used to inform all interested services of occasional and important facts. Some examples are error alarms and indications of arrival at specific points of the mission, etc.
3) Remote Invocation is an intuitive way of performing one-to-one modeling of interactions between services. Some examples are the activation and deactivation of actuators, or requesting some form of calculation to a service. Therefore, in addition to variables and events, the services can expose a set of functions that other services can invoke or call remotely.
4) The File Transmission primitive is basically used to transfer large sets of file-structured information from one node to another when there is a need for the safe transfer of XML configuration files, payload data, and mission-oriented data processed on board the UAS.

Service Oriented Architecture is becoming common in domains that can benefit from loose coupling among interacting services. This architecture provides an increase in the interoperability, flexibility, and extensibility of the designed system and of its individual services. Following this vision, we implement services that represent the different components and functionalities that make up the complete USAL architecture.

Middlewares permit the design of a complex service architecture by abstracting the designer from all inter-service communication details. However, this ease of development is confronted to a decrease of performance that may limit certain real-time operations. For this reason USAL employs two timing zones: one within USAL and another one between VAS and the autopilot. Within USAL the main driving event is the waypoint. Events are triggered because vehicles reach certain positions in space, for example, to activate/deactivate payload, or to decide which is the upcoming leg the UAS needs to fly. However, if the UAS flight needs to be continuously monitored, this operation needs to be implemented within the AP–VAS timing zone given the fact that their direct link (external to the SOA network) at most times will support hard real-time requirements.

IV. Virtual Autopilot System

The VAS belongs to the set of services defined in the USAL flight category. The VAS is specially designed to operate as an interface between the autopilot and the USAL. It is a system that at one end interacts with the selected autopilot and therefore needs to be adapted to its peculiarities. At the other end, it interacts with the other USAL services. The VAS operates similarly as drivers work on OSs, removing the implementation details from actual autopilot users. The VAS is a service that provides a standardized interface to the particular autopilot on board the UAS. For other services in the USAL, the VAS is a service provider that offers a number of information flows to be exploited by them. Given that not all autopilots are equal, the VAS follows a contract between it as a service provider and its potential clients. This means that all the information provided by this service is standardized independently of the actual autopilot being used.

The inclusion of the VAS greatly improves the flexibility of the USAL framework because of the following reasons:

1) the autopilot unit can be replaced by a new version or a different product, but this change will have no impact on the system except for the VAS; that is, it implements an abstraction layer in order to isolate the system of autopilot hardware changes;
2) an increased level of functionality is provided, permitting the operation with a virtually infinite number of waypoints, thus overcoming a limitation present in all UAS autopilots that have been studied;
3) commercial autopilots mainly focus on waypoint navigation, whereas UAS operation may require considering a global perspective, from takeoff, to the mission, and back to landing. The VAS promotes standardized mission-oriented states in order to cope with more elaborated operational requirements.

With these intention statements, the VAS fits perfectly within the USAL philosophy. Any specific autopilot interface, change or version update will not affect the UAS development effort. Also, the VAS greatly simplifies the integration of the autopilot within the USAL. In that way, USAL services will always interact with the same autopilot interface.

However, the VAS is not a replacement for the real autopilot. The VAS will simply provide either abstraction mechanisms to facilitate the way in which actual autopilot capabilities can be exploited; or increase the standardization of the AP interface (like providing a common waypoint definition and then translating it into the particular format

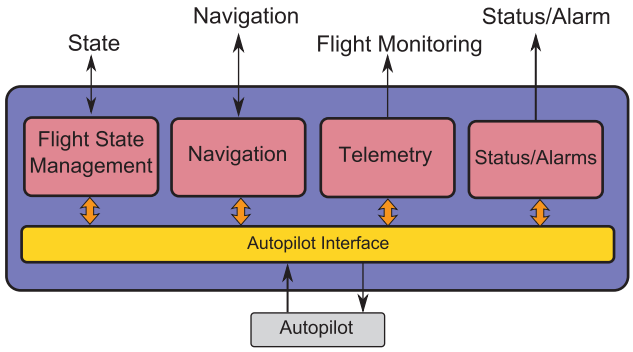

Fig. 5 Overview of the overall VAS architecture.

each autopilot uses). Highly specific autopilot functionalities may require additional nonuniform interfaces to be added, like special drivers permit accessing noncommon hardware in OSs. Future work will address this issue, for example, including a generic mechanism to activate user-defined autopilot modes and parameters.

The VAS is specially suited to work in conjunction with the FPMa and the FM. The FPMa is the USAL service responsible for processing and executing the proposed flight plan, but it does not operate in a stand-alone fashion. To execute the flight plan, the FPMa sends navigation commands to the VAS. These commands mainly consist of waypoints that the aircraft has to fly. Since the flight plan is specified in terms of legs, a certain translation process is needed to convert them into the waypoint sequences expected by the VAS. This flow of waypoint commands is the main form of interaction between the FPMa and the VAS.

The FPMa also issues other commands, for example, a cancelation command would be used for asking the VAS to ignore a number of waypoints and directly jump to a given leg, or when an emergency forces an alternative plan to be executed. Another type of command allows the FPMa to change the VAS operation mode to request special operations such as takeoff or landing.

The FM is the on-ground service that interacts with the VAS and keeps track of the mission (see Figs. 3 and 4). The FM provides the possibility for the PiC to configure and interact with the VAS. It shows a major schema of VAS states and their transitions. By means of this view, the PiC can change the VAS state. The FM is responsible for displaying all VAS generated telemetry to the PiC.

V. VAS Design

Virtual Autopilot System architecture has been divided into four main areas: Flight State Management, Navigation Information, Flight Telemetry, and Status/Alarm Information, as seen in Fig. 5.

The objective of the Flight State Management is to provide a uniform and well-structured model of operation, regardless of the underlying autopilot operational capabilities. Navigation Information will define the flows of information required to determine the path that the aircraft follows according to the selected operational mode. The Flight Telemetry area relates to the need to standardize a common set of attitude and position data extracted from the autopilot; that is, VAS enables the exploitation of the autopilot telemetry by other applications within the USAL. Finally, the Status/Alarm information gives information about the airworthiness of the autopilot and VAS. As shown in Fig. 5, Flight Telemetry and Status/Alarm information are outgoing information flows, whereas Navigation and State Management have an input/output flow. All these information streams will be described in detail in the following sections.

A. VAS Flight State Management

Most existing autopilots just focus on primitive navigation operations. In contrast, we have developed a set of mission oriented VAS states that focus on the overall operational sequence that a UAS should follow. The actual functionalities of the underlying autopilot are not determinant because the VAS will implement the missing high-level capabilities employing existing functionalities available within the autopilot (basically waypoint navigation and/or heading-based navigation). In this way, we ensure that the other services in the USAL always exploit a stable set of autopilot capabilities.

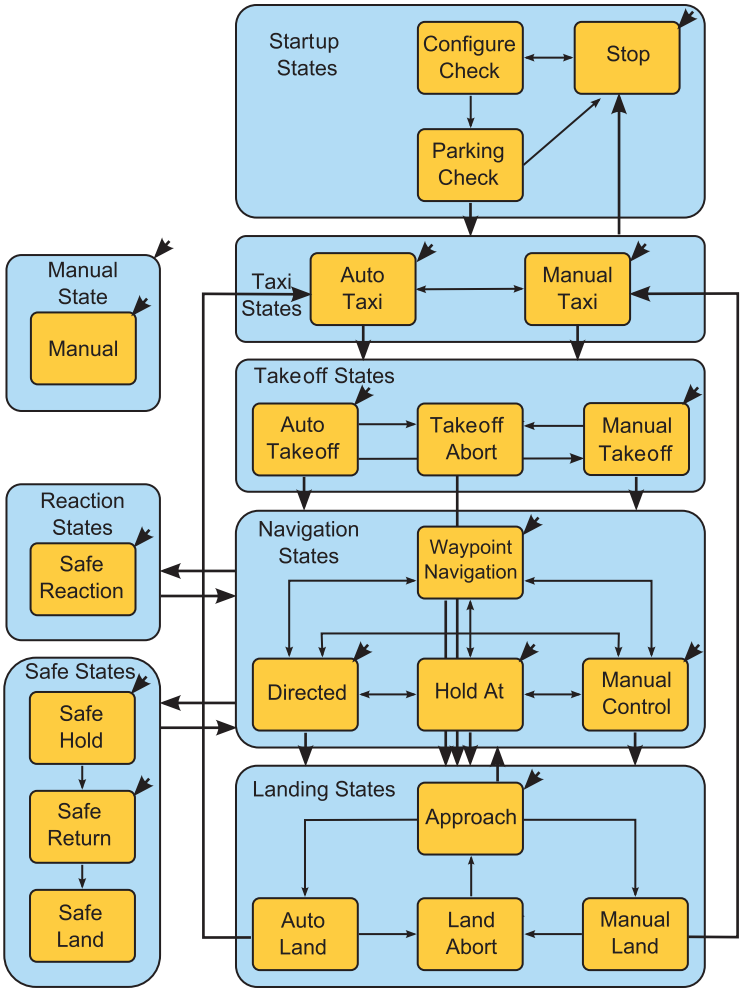

Fig. 6 Virtual Autopilot System state diagram. Arrows indicate allowed transitions.

The VAS organizes the operational states in groups of desired functions that include: the UAS startup and checks prior to flight, taxiing, takeoff, classical navigation, landing, and taxiing after landing. Also included are reaction states that allow conflict avoidance functions to take over standard operations; and safety states that provide a minimum set of autonomous contingency reactions. Each high-level function is then refined into specific states that implement the exact concept of operation. Within each state, the UAS develops specific tasks in order to achieve the operational requirements. Figure 6 illustrates the complete VAS state diagram and the allowed transitions between states. The initial state inside each group is shown with an arrow on the top right corner. When a given group of states is entered, only states marked as initial ones can be reached. The other arrows show the permitted transitions between states. The diagram begins with the UAS startup states at the top and, from there, the rest of the states that cover the overall operational sequence can be found in descending order. The FM employs a dedicated HMI (see Fig. 4) to help the PiC manage VAS operational workflow.

1. StartUp States

The group of startup states is related to the system initialization. It is composed of three states: Stop, Configure and Check, and Parking. Operations start at the Stop state. Only when the VAS and overall USAL services are properly configured, the UAS can change to the Parking state. In this state, the engine is switched on; then, sensors, servo

mechanisms, and communication modules will be checked again with the engine working. After the check list, the UAS can go on to taxi states. Next, we describe the states which make up the Startup group more in detail.

Stop State: On Startup, the VAS starts by default in the Stop state. In the Stop state, the overall USAL startsups and all commanded servos are disabled to avoid accidents.

Configure and Check: In this state, the VAS gets configured and executes several built-in self-test. The VAS configuration is described by XML files that contain the autopilot properties, the aircraft properties, and VAS settings for implementing the mission. All these operations are carried out with the UAS on the ground still inside a hangar or parking area.

Parking and Check: Once up and running, the VAS sets up all the connections with the underlying autopilot. Then, it starts to publish all the autopilot streams to the other USAL services; e.g., the telemetry flow is activated and UAS positioning is validated. During the Parking state, a preflight checklist is carried out with the UAS engine eventually working. This task is commanded from the FM where the PiC is controlling the UAS.

2. Taxi States

Now the UAS can proceed to the designated runway in order to start the flight. The VAS can support this operation in two different ways: Auto Taxi and Manual Taxi. With the Auto Taxi, a specialized auxiliary service will generate ground navigation commands to reach the runway. In Manual Taxi, the PiC uses the FM to drive to the correct runway holding point by the correct taxiway (see Fig. 4 for an outline of the dedicated HMI). It is the pilot's responsibility to do this in accordance with the taxi procedures and the runway in use. The VAS will set up certain built-in limits according to the actual ground speed and steering of the UAS. Switching from auto to manual is allowed at any time, whereas the converse is only allowed if the UAS is properly aligned within the taxi route.

Current autopilots (see Sec. II) do not provide an auto taxi capability. In these cases, the VAS foresees a complementary service that will have to implement the taxi algorithm for guiding the UAS to the desired runway. On the other hand, if anything goes wrong, the VAS can give the control to the PiC to change to Manual Taxi. Then, when the UAS has reached the runway holding point, it can transition to the takeoff states.

Table 1 illustrates some Taxi state parameters required to support the Taxi operation safely. "MaxTaxiSpeed" defines the maximum speed over the taxiway, whereas "UsageTaxiThrottle" limits the maximum throttle usage in order to avoid sudden accelerations. Limitations are employed to limit other aspects like: the turn radius, speed while turning, etc.

Taxiing is a complex operation, especially in automatic navigation. In noncontrolled airports, it is the responsibility of the PiC to ensure a safe Taxi. However, in controlled airports, this operation may be commanded to some extent by the Tower Controller. Usually, the number of taxiways in an airport are considerable. Therefore, in order to select the correct taxiway, the FM provides special HMI interfaces [27] designed to help the PiC in order to achieve this goal (see Fig. 4).

3. Takeoff States

As in the Taxi, the takeoff states can also be carried out manually (Manual takeoff) or automatically (Auto takeoff). If the takeoff is carried out automatically and any eventual contingency occurs, the PiC can switch over to Manual takeoff to take charge of the situation. In case of performing a manual takeoff, the VAS will remain in this state until it can change to the Navigation states. Normally, this transition between states has to be done at a safe altitude. The selection of the Takeoff Abort state means that for some reason the UAS cannot change to Navigation states. In this case, the UAS joins an aerodrome traffic pattern and then it switches to Land states. In this way, the VAS assures a safe landing of the UAS if a problem has been detected during the takeoff procedure.

The Takeoff control parameters are shown in Table 2. "TakeoffSecureAltitude" indicates the minimum safe altitude where the initial takeoff climb finishes and the VAS commands a turn into the end of departure waypoint (EDWP). By

Table 1 General parameters to the taxi states

Taxi parameters name	Unit	Description
MaxTaxiSpeed	m/s	Maximum taxi speed
UsageTaxiThrottle	Percentage	Percentage of throttle usage

Table 2 General parameters to the takeoff states

Takeoff parameters name	Unit	Description
TakeoffSecureAltitude	Meters	Altitude where finishing the takeoff state
TakeoffVerticalSpeed	m/s	Rate of descent or climb along the takeoff state
TakeoffSpeed	m/s	Speed over the runway before climb
TakeoffDecisionPoint	Wp object	Runway threshold to abort the takeoff

default, this altitude is 150 m, however, some airfields may vary altitude to impose other limits due to geographical constraints. "TakeoffVerticalSpeed" sets up the instantaneous climb rate. "TakeoffSpeed" is the threshold speed before rotation should be performed. "TakeoffDecisionPoint" describes the runway threshold decision point where the takeoff can be aborted. This point is implemented in order to be able to automatically abort a takeoff in the event of the expected acceleration and speed parameters not being reached.

Manual Takeoff: Assuming that the UAS is in the proper runway holding point, the VAS accepts commands from the PiC in order to implement the Manual Takeoff operation. Full throttle control is offered to the PiC, although limited steering control is available in order to avoid an improper operation. As soon as a safe speed and altitude are reached, the VAS transits into Navigation states. Speeds and altitudes can be parameterized as desired depending on the details of the UAS (see Table 2).

Auto Takeoff: If the autopilot is capable of performing an automatic takeoff, the VAS lets the autopilot execute this initial part of the procedure. If this is not the case, the VAS implements the execution of this state using the available autopilot control primitives; that is, the VAS can either command the autopilot to perform a built-in takeoff operation or implement the automatic takeoff extending the autopilot capabilities [26,28].

Two scenarios exists when performing a UAS takeoff operation. There is the case when the runway in use has a Standard Instrumental Departure (SID) published, and the UAS has been cleared to execute it. In this situation, the takeoff procedure will consist of just the takeoff roll and an initial climb out to transition immediately to the Navigation states and perform the published SID externally commanded by the FPMa.

In case there are no SIDs published, the VAS, together with the FPMa and FM, proposes a specific VFR-like procedure consisting on an initial climb and a transition to an EDWP. This EDWP is the link between this departure procedure and the beginning of the mission flight plan. Therefore, at the EDWP the UAS starts the Navigation states commanded by the PiC or by the FPMa. Figure 7 shows a generic example of a runway with five associated EDWPs which are automatically generated, but can always be edited or deleted by the PiC during the preflight activities.

Depending on the initial waypoint of the flight plan, a unique EDWP will be selected from the five EDWPs associated with five different areas (A, B, B′, C, C′) that may contain this initial waypoint.

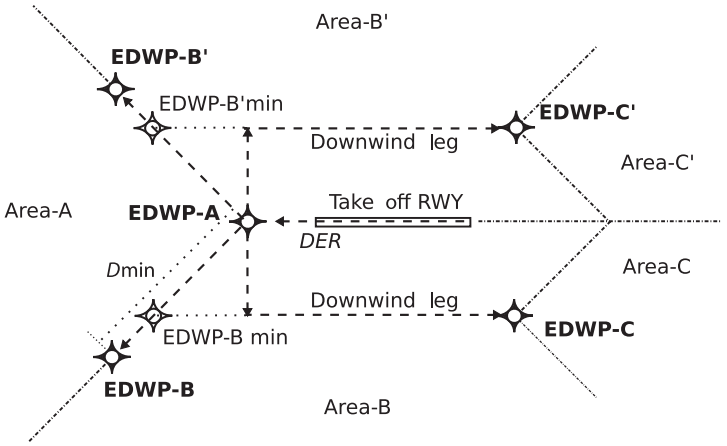

Fig. 7 Virtual Autopilot System definition of EDWPs.

Table 3 General parameters to the navigation states

Navigation parameter name	Unit	Description
NavigationMaxAltitude	Meters	Maximum navigation altitude
NavigationMinAltitude	Meters	Minimum navigation altitude
NavigationMaxSpeed	m/s	Maximum navigation speed
NavigationMinSpeed	m/s	Minimum navigation speed

4. Navigation States

When the UAS has reached an EDWP, it changes automatically to Navigation states that are composed by Waypoint Navigation, Directed, Hold-At, and Manual Control states. In Waypoint Navigation, the UAS follows the waypoints that are provided to the VAS. In the Directed state, it will simply maintain a specific altitude, airspeed, and heading. In the Hold-At state, it executes a holding pattern around an indicated waypoint. Finally, in the Manual Control state, the PiC has a supervised control over the vehicle to maintain a certain safety envelop.

Table 3 shows the parameters which can be managed during the Navigation states. These parameters define the speed and altitude limits to all Navigation states. They are not static parameters during the entire mission, they can be configured from one leg to another one depending on the mission and safety requirements.

Waypoint Navigation: The VAS offers two different mechanisms for operation in the Waypoint Navigation state: static and streamed Waypoint Navigation.

Static Waypoint Navigation is commanded by the PiC through the FM (as seen in Fig. 4). In this case, the VAS sequentially flies the waypoints stored in the static waypoint vector where they have been previously uploaded.

In streaming Waypoint Navigation, the VAS reads a continuous stream of waypoints generally generated by the FPMa. The waypoints are stored in an internal queue to be sequentially processed in the same order that they are inserted. Once a waypoint is flown, it is discarded. It is competence of the FPMa to generate and validate each required waypoint, and their sequence, according to the characteristics of the selected flight plan.

Directed: The VAS keeps the UAS under stabilized flight, allowing a restricted number of parameters to be modified: heading, speed, and altitude. For safety reasons, there is a timeout for staying in the Directed state with the same heading. If the specified time is exceeded, the VAS will switch to the Safe Hold state.

Hold At: The VAS maintains the UAS under a stabilized holding pattern characterized by a number of parameters that can be trimmed to achieve the desired holding structure (see Fig. 8). This state is defined by an Inbound track, a Holding Fix, a turning direction, and two time parameters: t1 and t2. Essentially, the UAS flies along the Inbound Track until the Holding Fix is reached, and then flies a right/left turn rectangular pattern defined by three additional waypoints. Again, speed and altitude can also be modified as usual through the Input Navigation parameters.

Table 4 depicts the Hold At state parameters. "NewT1HoldTime" and "NewT2HoldTime" define the duration of the holding pattern. With "NewSetHoldFix", we define the waypoint over which the UAS starts its first turn. We also define "NewSetHoldDirection" in order to choose the turn direction of the holding pattern. Finally, "NewSetHoldAtFix" sets up the holding point around which the UAS will turn. Therefore, we have two ways of specifying the holding

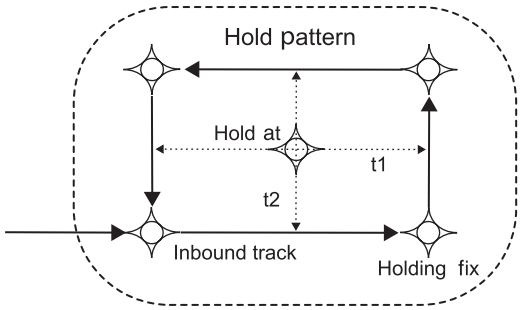

Fig. 8 Hold at pattern state.

Table 4 Active input navigation packets during hold at state

Input hold at parameters	Units	Description
NewT1HoldTime	Seconds	Duration of the horizontal track
NewT2HoldTime	Seconds	Duration of the vertical track
NewSetHoldFix	N/A	Set up current fix for holding operation
NewSetHoldSide	N/A	Set up current side for holding operation
NewSetHoldAtFix	N/A	Set up the center waypoint of the holding operation

pattern: one is by managing the center of the hold and the other is by controlling the waypoint over which the UAS starts the procedure.

Manual Control: The UAS can be operated manually by means of a joystick connected to the FM. The PiC has direct control over the UAS, but under similar restrictions as in the Navigation states; i.e., max/min altitude, max/min speed, max/min attack angle, etc. These parameters help the pilot avoid any critical or radical procedures. If the pilot wants to operate without any restrictions, it has to change to the Manual state in which the UAS is outside Navigation state boundaries.

5. Landing States

Along with the Takeoff states, the Landing states address one of the most important issues that will arise if extensive civil UAS application become a reality in the near future, in a scenario where manned aircraft will coexist with unmanned vehicles. Like in the Auto Takeoff state, we may have Standard Terminal Arrival Routes and/or Instrumental Approach Procedures published for the runway in use. In this case, this approach phase is assumed in the Navigation states.

In the case of not having an instrumental approach available, the VAS proposes a specific VFR-like procedure detailed in [26]. Briefly, the UAS will overfly the airfield at a height greater than the highest of the aerodrome traffic patterns. A hold will take place at a point over the aerodrome's vertical. At this point, the aircraft will be able to wait at a safe altitude before joining the aerodrome traffic pattern. Once in the holding, the PiC will select the traffic pattern or the Air Traffic Controller (ATC) will command the integration in one defined direction. Then, a holding Exit Waypoint is defined along with an Integration Waypoint and the Initial downwind Waypoint. Then, the UAS will fly the upwind and crosswind legs and integrate the downwind leg at the height of the circuit. After this integration, the UAS will follow the standard traffic pattern with a downwind, base, and final legs. Once in the final leg, the VAS will transition to Auto Land or Manual Land states (see Fig. 9). If any problem occurs during the operation, the selection of the Land Abort state will make the UAS climb up to a safe altitude and reinsert it in the approach pattern

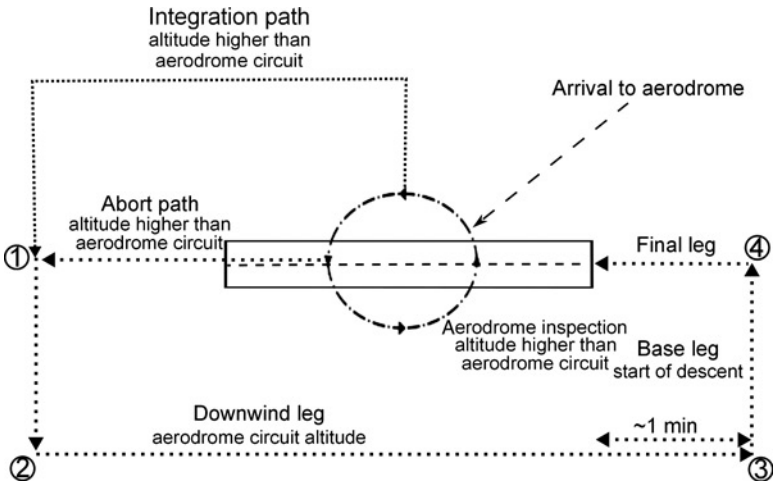

Fig. 9 Overview of land pattern state.

Table 5 Active input navigation packets during approach state

Input land pattern packets	Units	Description
CrossWindTime	Seconds	Set up the crossWind time in the Approach
DownWindTime	Seconds	Set up the downWind time in the Approach
InVerticalHoldAltitude	Meters	Set up current altitude for holding operation in the runway
InVerticalHoldSpeed	m/s	Set up current speed for holding operation in the runway
InVerticalHoldAt	Wp object	Set up wp. of the center for holding operation in the runway
TrafficSpeed	m/s	Set up the speed of the traffic circuit operation
TrafficAltitude	Meters	Set up the altitude of the traffic circuit operation
ApproachMode	N/A	Describes the land pattern mode

(through the Approach state). Once the UAS slows down on the runway, the VAS will switch to the Taxi state again to finish the operation.

Approach: The VAS employs a number of parameters that permit to define the details of the circuit performed around the airfield. These parameters specify the fly times required to define the initial holding pattern (initial altitude, speed, duration, and center point for the maneuver). The actual circuit is defined by CrossWind and DownWind parameters, as well as all the details on the glide slopes and the runway itself. Additionally, traffic speed and altitude designed for the characteristics of each individual UAS are required.

Table 5 shows the parameters of the Approach state. Combining the "CrossWindTime", "DownWindTime", and the specific runway information, the VAS establishes the different times associated to each traffic leg. Along with this information, we need the desired traffic speed and altitude, defined in "TrafficSpeed" and "TrafficAltitude". The parameters that describe the holding pattern vertical to the runway are: "InVerticalHoldAltitude", "InVerticalHoldSpeed", and "InVerticalHoldAt". The first two parameters fix the holding altitude and the speed. The last parameter defines the coordinates of the center of the holding racetrack.

Auto Land: As in the Takeoff state, the VAS can also provide this procedure on top of the commercial autopilot being used. If the autopilot does not support this feature, the VAS can implement it. To develop a safety procedure, autopilot behavior at low speed and low altitude has to be sufficiently correct. Some autopilots lose efficiency in this situation, and hence autopilot precision has to be accurate enough to allow implementation of a proper landing maneuver.

Manual Land: In the Manual Land state, the VAS grants the UAS control to the PiC through the FM. The pilot is expected to generally follow the same landing procedure specified for the automatic landing process. The VAS continuously monitors the operation to guarantee that the UAS is kept under a minimum landing envelope. If outside this envelope, the VAS immediately reverts to the Land Abort state. This functionality can be overridden if a nonconventional landing is mandatory.

Land Abort: When the UAS is in the Auto Land state, the VAS computes the difference between the Desired Touch-Down Fix and the real one. If the deviation obtained is greater than a predefined threshold, the missing approach procedure should start by passing to the Abort Land state. In future versions of the VAS, it is clear that the same abort procedure should be used with respect to any lateral deviation, speed variations out of a valid range, or rates of descent out of valid margins. In the event of an aborted landing, the UAS should fly until rejoining the traffic pattern of the runway.

6. Safe States

If any failure occurs during the Navigation states (basically the VAS loses contact with the FM due to a data link failure or an internal architecture failure), the VAS switches to the Safe states. These states are composed of the Safe Hold, Safe Return, and Safe Land states. When a failure is detected, the VAS changes to the Safe Hold. The UAS will remain in this state while recovering from failure. After a timeout, the UAS will switch to the Safe Return state. In this state, the UAS will return to the predefined emergency runway in order to start the Safe Land state employing similar parameters to those used for an standard landing operation.

Note that the USAL architecture itself contains a contingency module that may trigger an emergency landing procedure in which more elaborated strategies are defined [25]. For example, the USAL can internally update the

Table 6 Active input navigation packets during the safe hold state

Input safe hold packets	Units	Description
newSafeHoldTime	Seconds	Time to stay in hold state
newHorizontalSafeHoldTime	Seconds	Duration of the horizontal track
newVerticalSafeHoldTime	Seconds	Duration of the vertical track

exact emergency landing site to be used at each phase of the flight plan evolution. However, in case the USAL fails, the VAS provides similar capabilities but employing simplified decision strategies.

Safe Hold: This state has been designed to offer a waiting pattern to recover from minor contingencies without fully aborting the operation. These contingencies can become a real problem if they persist for a long time. The VAS will remain in this state for a predefined amount of time, so that either the Pic or the UAS itself has the opportunity to fix the problem before aborting the mission.

The parameters in Table 6 describe the holding pattern and the timeout to stay in this state before it changes to the Safe Return state.

Safe Return and Safe Land: After a timeout in the Safe Hold state, the VAS will switch to the Safe Return state. This means that there has been no successful response to the contingency. Therefore, the VAS follows the safe protocol to recover the UAS. In this case, the UAS returns home in the Directed mode to land as soon as possible. When the UAS is arriving at the airport, it overrides the Safe Land mode. In this case, the VAS skips the Land Pattern and commands the UAS to land in the selected runway.

7. Safe Reaction

There are several conflict avoidance situations that need an immediate reaction from the UAS. As in manned aircraft, avoidance can be divided into: conflict avoidance when enough lead time exists to prevent the conflict with minor flight plan modifications; or collision avoidance when an immediate radical maneuver is necessary to avoid a mid-air collision. Although conflict avoidance is coupled to the FPMa, the VAS retains a specific state to respond to these immediate requests. Switching into safe reaction means stopping the current VAS operation to react as soon as possible with a catalog of preselected maneuvers. This reaction will be commanded by an external service (within the awareness services in USAL) that may request a sequence of operations to guarantee that the collision is avoided.

In Safe Reaction state, the VAS accepts several parameters to configure the reaction to conflicts. Parameters include the heading to avoid the obstacle, the ascend/descend angle and the reaction priority. By dynamically changing the ascend/descend angle, TCAS-like reverse maneuvers can be implemented.

Through the priority parameter the VAS knows how radical the maneuvers have to be. The operation is managed through the "NewSafeReactionHeading" and "NewSafeReactionPriority" parameters, although other parameters to perform vertical maneuvers may be implemented (Table 7).

B. VAS Navigation Information

In the USAL framework, the FPMa is in charge of generating and managing the navigation commands sent to the VAS (see details about FPMa in [23]). The VAS navigation information is composed of two sets of information: Output Navigation and Input Navigation information. The aim of Output Navigation is to publish the UAS navigation status to the rest of the services. Input Navigation provides the autopilot with navigation primitives that determine the flight path.

Table 7 Active input navigation packets during the safe reaction state

Input safe reaction packets	Units	Description
NewSafeReactionHeading	Radians	Set up current heading to evade an obstacle
NewSafeReactionPriority	Integer	Set up current operation priority

Table 8 Output navigation information by the VAS

Protocol primitive	Name	Composition	Unit	Description
Event, function	currentWp	Lat., lon., alt.	Rad., meters	Waypoint information
		Identifier	N/A	where the UAS goes
Function	previousWp	Lat., lon., alt.	Rad., meters	Previous waypoint info
		Identifier	N/A	
Function	rwySituation	Lat., lon., alt.	Rad., meters	Runway info
		Heading	Radians	from the
		Length	Meters	autopilot
Function	altRwySituation	Lat., lon., alt.	Rad., meters	VAS alternative
		Heading	Radians	Runway information
		Length	Meters	
Function	uavDirection	TAS	m/s	Current target TAS,
		Altitude	Meters	altitude,
		Bearing	Radians	Bearing and heading
		Heading	Radians	
Function, Event	vasState	State	N/A	Current VAS state

1. Output Navigation Information

This information basically defines where the UAS is going at any moment, in which direction it is moving, and which waypoint it is flying (see Table 8).

Almost all output VAS navigation primitives have a semantic of remote invocation. This means that when a service in the network needs any of these data, it can request them remotely. However, messages such as current waypoint "currentWp" and VAS state "vasState" are also published as an event. These data are essential to know the current position and phase of the mission. Services such as FPMa or FM need to know this information at regular time intervals. Given that this information is important to the UAS flight, we must guarantee the reception of the information sent to all the subscribed services; therefore, we use MAREA middleware events. The "previousWp" message indicates the last flown waypoint. "rwySituation" and "altRwySituation" indicate the position of the selected runway for nominal operations and alternative operations (such as fuel or battery contingencies), respectively. Finally, "uavDirection" indicates the current track that the UAS is flying. The true airspeed, altitude, and heading are also included in this message.

2. Input Navigation Information

The next group of information is the Input Navigation information. This basically defines VAS configuration parameters and commands for the autopilot operation, as well as several parameters that define the UAS navigation according to the selected VAS state (see Table 9).

To set the target airspeed, the altitude, and heading during the mission, the VAS provides "newUavSpeed", "newUavAltitude", and "newHeading" messages, respectively. This information only takes effect once in the navigation state. Through the "newMainRwy" and "newAltRwy" messages, the VAS offers the possibility of changing the coordinates of the main and the alternative runways. With the "changeVasState" message, services such as the FPMa, FM, or awareness services can switch the desired VAS state. This message is composed of VAS state type and the state parameters. With the first field, we specify the new VAS state transition. With the second one, we define the parameters of each state. Finally, the VAS provides three messages in order to manage the mission waypoints: "clearWps", "skipWp", and "discardWps". The first message is used to clear all the flight plan waypoints. The second one is needed to skip a single waypoint instead of the entire flight plan; and with the last message the VAS can discard a range of flight plan waypoints.

With the "maxTimeMission" message, the VAS sets up the operation time limit. If this limit is surpassed, an alarm will be raised. The servos' deflection may be directly managed with the "deflecSurfCont" message. This information sets the data that will act directly on the aileron, elevator, rudder, and throttle servos. To feed the autopilot with the mission waypoints, the "newWp" message is used.

Table 9 Input navigation information by VAS

Protocol primitive	Name	Composition	Unit	Description
Event	qnhGround	Pressure	Pascals	Sets the QNH pressure for the altimeter
Event	gndLevel	Ground level alt.	Meters	Set the ground level alt. to the autopilot
Event	maxTimeMission	Time	ms	Set mission time
Event	deflecSurfCont	Aileron, rudder, elevator, throttle	Radians	Direct surface control packet
Event	newWp	Lat., lon., alt.	Rad., meters	Read waypoint information
		Speed	m/s	Where the UAS goes
		Fly over	N/A	
		Identifier	N/A	
Event	newUavSpeed	IAS	m/s	Set indicated air speed
Event	newUavAltitude	Altitude	Meters	Set UAS altitude
Event	newBearing	Direction	Radians	Set UAS bearing
	newMainRwy	Lat., lon., alt.	Rad., meters	Set the coordinates
Function	or	Altitude	Meters	of the main
	newAltRwy	Heading	Radians	and alt. Runway
		Length	Meters	
Event	changeVasState	State type and param.	N/A	Set the VAS state
Event	clearWps	Event	N/A	Clear flight plan waypoints
Event	skipWp	Waypoint identifier	N/A	Skip one waypoint to another
Event	discardWps	Waypoints identifier	N/A	Discard range of waypoints of
		Range		flight plan

C. VAS Flight Telemetry

There are several ways to display the information generated by the sensors of the autopilot. Usually, autopilot manufacturers group all this information in large streams of data, which are sent via a radio modem at a certain frequency. The VAS offers this information over a LAN on board the UAS to the rest of the services. The information is semantically grouped in such a way that this information relates to parameters, situations, or attitudes of the aircraft, independently of the real autopilot hardware and sensors (see Table 10).

The instantaneous angular position, acceleration vector, speed vector, and rate of turn of the UAS are indicated by the "uavAngles", "uavAcceleration", "uavSpeed", and "uavRateTurn" messages, respectively. The instantaneous coordinates of the UAS is provided by the "uavPosition" message. The Indicated Airspeed (IAS) and the True Airpeed (TAS) are offered though the "uavAirSpeed" message. In addition, the estimated wind direction around the UAS and the mission time in seconds from VAS startup is published with the "windEstimated" and "missionTime".

Table 10 Flight management information published by the VAS

Protocol primitives	Name	Composition	Unit	Description
Variable	uavAngles	Roll, pitch, yaw	Radians	Roll, pitch and yaw angles
Variable	uavAcceleration	X,Y,Z	m/s^2	Acceleration in UAS X,Y, Z axis
Variable	uavRateTurn	X,Y,Y	rad/s	Rate of turn in UAS X, Y, Z axis
Variable	uavPosition	Lat., lon., alt.	Radians and meters	3D UAS position
Variable	uavSpeed	North, east, down	m/s	3D speed in the UAS
Variable	uavAirSpeed	IAS and TAS	m/s	Air speed data in UAS
Variable	windEstimated	North, east, down	m/s	3D wind speed estimated
Variable	missionTime	Time	ms	Mission duration
Variable	uavSurfaceControl	Alieron, rudder, flaps, elevator	Radians	Surface control positions

Table 11 Status and alarm information

Protocol primitive	Name	Description
Event	gpsApAlarm	GPS alarm
Event	outRangeTempApAlarm	Temperature outside range
Event	accApAlarm	Acceleration alarm
Event	rateTurnApAlarm	Rate of turn alarm
Event	ImuApAlarm	IMU alarm
Event	magnetometerApAlarm	Magnetometer alarm
Event	pressureAltimeterApAlarm	Pressure altimeter alarm
Event	anemometerApAlarm	Anemometer alarm
Event	missionAlarm	Mission time alarm
Event	wpRangeAlarm	Waypoint range alarm
Event	wpProcessAlarm	Error processing parameters
Event	lackMainRwy	No main runway
Event	speedAlarm	Speed range

In general, UAS autopilots provide a great deal of information from all the sensors; however, only a small portion is of real use when implementing a system oriented toward mission development. The Flight Monitoring Information has been chosen taking into consideration which information could be useful in mission-related terms.

D. VAS Status/Alarm Information

An autopilot is a complex piece of hardware that needs constant monitoring. With Status/Alarm information, the VAS publishes the autopilot and VAS status. When any part of these devices fails, the VAS sends an alarm to the network (see Table 11).

There are certain defined alarms that control the links between the GPS receiver and the satellites (GPS Autopilot Alarm). The VAS also provides detailed alarms for notifying problems in the different sensors within the autopilot (Accelerometer, Gyroscope, Magnetometer, Anemometer, and Pressure altimeter). However, operation of the UAS can usually continue in a degraded mode. In this case, the UAS will usually try to return to base for maintenance. Some alarms manage the specific operation of VAS service. The Waypoint Range alarm will be raised when the FPMa (or the service in charge of providing the waypoint list to the autopilot) tries to feed a waypoint that is not coherent with the rest of the flight plan (too far away from the previous waypoint). This usually indicates a problem with the FPMa or the flight plan itself.

Another specific situation that is notified by the VAS is when a main runway has not been uploaded. Finally, in the event of an internal error the VAS will raise the Process Error alarm, indicating that it has detected some abnormal behavior in its internal calculations.

VI. VAS Implementation

In this section, the proposed design for implementing the VAS is outlined. The VAS uses a two-level design: one that interacts with the USAL [called the Virtual Autopilot Layer (VAL)], and another that interacts with the autopilot [called the Autopilot Layer (APL)]. Both pieces of software are defined as "layers" (see Fig. 10). The VAL is in charge of interacting with the USAL. It defines the VAS API in order to provide the same abstracted autopilot interaction for the remaining USAL services. This layer never changes, irrespective of which autopilot the VAS is working with.

The APL is responsible for dealing with each particular autopilot. Here is where all the specific functionalities, language, formats, and behavior of the autopilot are dealt with. This layer has to accomplish certain requirements defined by the VAL. If the autopilot is changed, the only code that must be partially replaced is in the APL.

The VAL–APL abstraction is implemented through the use of the "IAutopilot" interface. The interface works as a contract carried out by the two layers and defines the compulsory methods that every autopilot class must have in order to be used by the VAS. When the autopilot is changed, it should have an associated autopilot class that fulfills all the requirements made by the interface. This is why interfaces work as contracts: every autopilot class must accomplish the obligations required by the VAS, through the interface, in order to work with them.

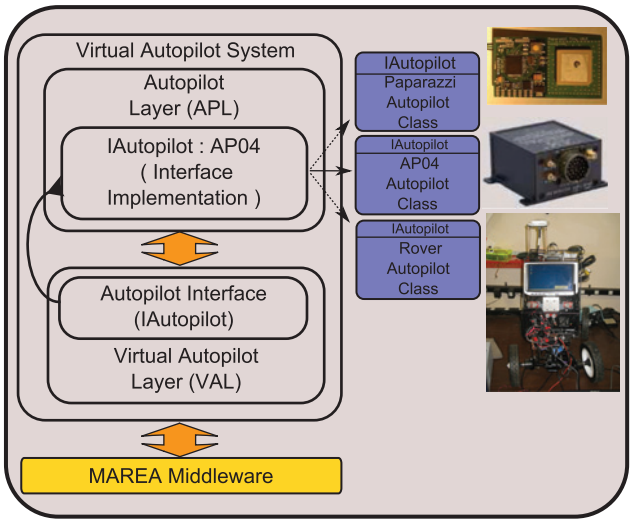

Fig. 10 Virtual Autopilot System interface composition.

To test VAS development, we have implemented the IAutopilot interface for three autopilots: Paparazzi, AP04, and a rover autopilot (see Fig. 10). To carry out these autopilot changes, we only have to develop a class that implements the complete set of methods required by the IAutopilot interface.

A. Autopilot Layer

This layer is responsible for adapting VAS to the peculiarities of each autopilot. It is composed of classes designed in a generic way to reduce the reconfiguration time which is one of the objectives of the USAL. Figure 11 shows the overall APL architecture.

The main class of the APL is the "*autopilot*" that implements the "IAutopilot" interface. This class acts as a wrapper that invokes other specific classes that manage aspects like the communications links with the autopilot and the VAL, or classes that manage the information flows for each type of data employed within the VAS. At the other end, the APL connection class manages all the specific characteristics of the direct connection between the autopilot and the VAS (physical transport, format of the messaging protocol, etc.).

The APL implements a global area class designed to share variables from different parts of the layer. This class works as a repository box where all flight information is stored. The most relevant components of this shared class are the following:

1) *Navigation Information:* The navigation information part formats and forwards the navigation USAL messages received by the VAS in its own specific language.
2) *States Information:* In the states information part, the APL receives a USAL state command. This command expects a concrete behavior. If the autopilot has no similar state, the APL layer has to implement it.
3) *Alarm Information:* The APL has to be aware of the alarms the VAS sends to the system. Therefore, if an alarm is not given by the autopilot, for example, that a waypoint is not valid, this part has to create this information and send it.
4) *Telemetry Information:* Most of the telemetry given by autopilots is similar. The duty of the APL here is to format and split the information given by the autopilot to the USAL telemetry packets defined in Tables 8 and 10.

As may be seen in Fig. 11, the navigation and states parts are an up-link stream. The information given by the VAL reaches the APL, and it has to be formatted, or created, in order to be sent to the autopilot, while the telemetry and alarm works the other way around. The information is taken from the autopilot and the APL should transform it into a suitable form for sharing it with the rest of the system.

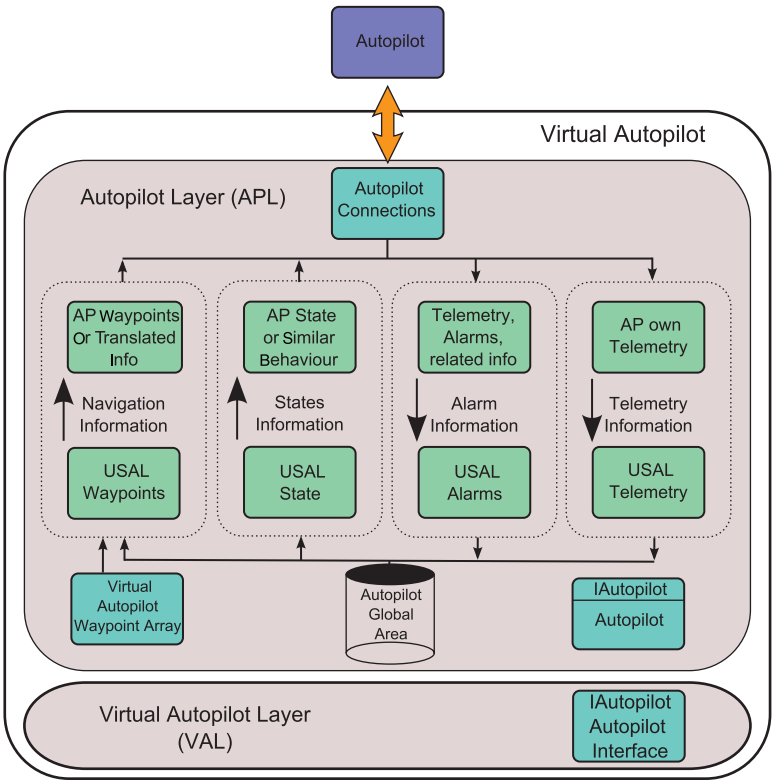

Fig. 11 Composition of APL.

Waypoint management is one of the most complex implementation issues. The APL implements a waypoint array class that creates a replica from a real autopilot waypoint array. When adding, erasing, or performing any kind of manipulation in the queue, this needs to be done locally. Once a waypoint is added or erased from the replicated buffer, this buffer sends the information to the autopilot in its own specific language.

B. Virtual APL

The VAL layer is composed of several classes described in Fig. 12. The VAL interfaces on one side with the APL layer and on the other side with the whole USAL architecture through a middleware. The *IAutopilot* interface for the APL is defined in the VAL, however, its implementation is always developed in the APL. As mentioned previously, the VAL will never change and the interface definition is the same for all the autopilots, although its implementation may differ.

To correctly operate as a USAL service, the VAS has to honor the *IService* interface declared in the MAREA middleware. This task is divided into two separate modules seen at the bottom of Fig. 12. The *Virtual Autopilot Service* is in charge of providing the overall interface compatibility when handling incoming events, functions, and parameterization files. All this message flow is redirected to the specific component within the VAL.

On the other side, the *Virtual Autopilot Publisher* collects all outgoing communication requests from inside the VAL and publishes them through the middleware. Additionally, this publisher is in charge of gathering and publishing all telemetry follows that are offered at constant rates, thus reducing the computational load of the remaining components. Like in the APL, the VAL employs a *Global Area* class to collect and centralize all relevant information. The publisher can take all the fixed rate outgoing data from this information pool without disturbing other components.

In addition to the aforementioned interfaces, the VAL is structured into four main components: State Management, Telemetry Management, Navigation Management, and Configuration Management.

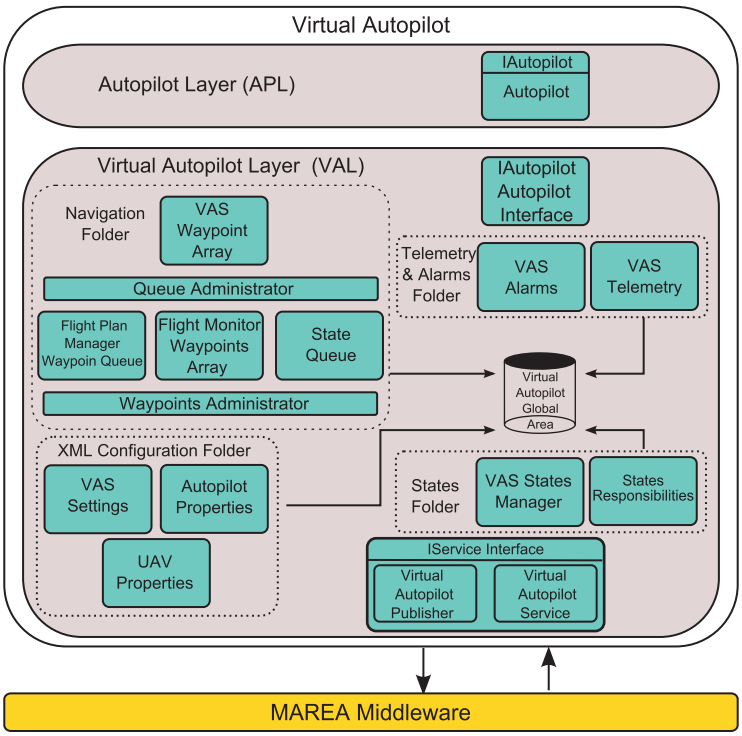

Fig. 12 Composition of VAL.

To parameterize some important aspects related to VAS configuration, a number of *XML Configuration* files are used. The information stored in these documents may vary depending on the mission, airframe, or autopilot. To maintain reconfigurability when the system is at the Configure and Check state, the VAS will consult these parameters and load them on internal data structures.

The *Telemetry and Alarms* manages the collection of the incoming autopilot telemetry and stores information related to the autopilot and overall system status. It also reports on any alarm or any anomaly detected in the system.

Within the USAL, different services are in charge of sending navigation commands to the VAS: the FPMa for complex flight plan management or FM for direct PiC management, and the Awareness services in case an avoidance reaction is necessary. In all cases, these commands take the form of new waypoints or direction changes. All these commands are collectively processed by the *Navigation* component, which is responsible for managing the internal waypoints queues that are later going to be sent to the actual autopilot.

Finally, the *State Management* component is responsible for implementing the operation state diagram, processing the events and conditions that produce transitions between states, and preventing forbidden transitions.

Figure 13 presents a detailed extension of the Navigation Management component located in Fig. 12. It shows all the internal blocks that implement the waypoint flow in the VAL. As can be seen at the bottom of the figure, there are different services that can generate waypoints and send them to the VAS: FM, FPMa, and the Awareness services. Waypoints can also be internally generated at certain states of the VAS itself to implement specific functionalities (like VAS land pattern). The VAL contains different data structures that store waypoint information: in particular, the FPMa Waypoint Queue, the FM Waypoint Array, the Awareness Waypoint Queue, and one array for each VAS state that needs to generate its own internal waypoints. In the figure, the Hold Waypoint Array is displayed as an example.

When a waypoint is received by the VAS, the Waypoint Administrator is in charge of analyzing the waypoint identifier and saving it in the correct queue or array. At the top of Fig. 13, the VAS waypoint array, which constantly maintains three waypoint positions, is depicted. This array is used to ensure that an exact synchronization exists between the autopilot internal waypoints and the mapped VAS waypoints.

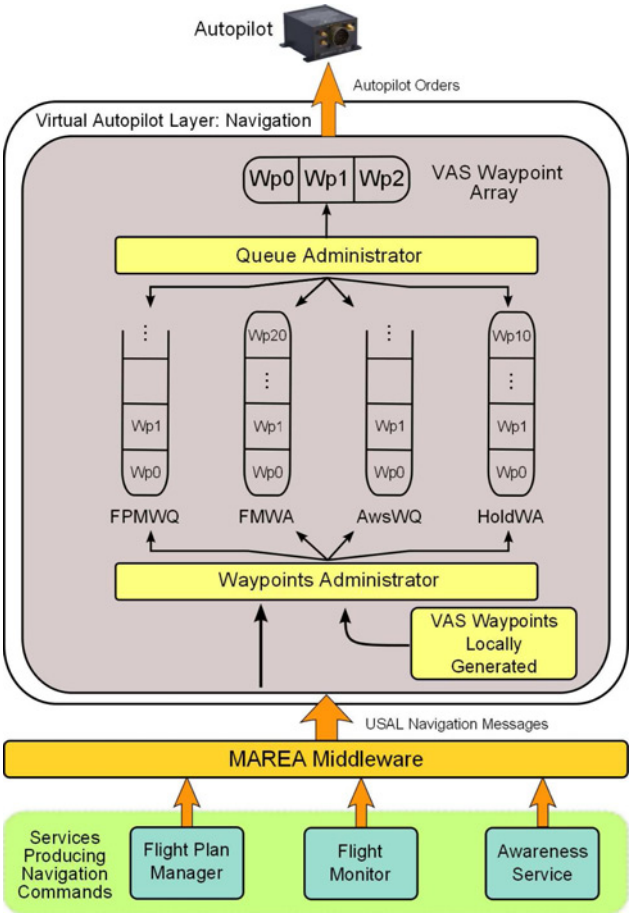

Fig. 13 Virtual Autopilot Layer navigation class.

VII. Experimental Results

The VAS has been implemented for three different autopilots under simulated environments: the commercial AP04 autopilot for both airplanes and helicopters from *UAV-NAVIGATION*, the open-source Paparazzi autopilot from ENAC for small UAS, and an autopilot for unmanned ground vehicles called Reflection Autopilot from NASA Ames Research Center [29,30] (see Fig. 14). These adaptations have been developed following and implementing the "IAutopilot" interface as described in the previous section.

Developing the VAS for the AP04 autopilot took about 28 weeks. In a first step, a whole AP04 interface was designed employing the open-source flightgear simulator as the underlying dynamic model. This allowed to test almost all critical functionalities before moving on to test the VAS on the real AP04 hardware. This period of time was divided in three different parts: The APL development, VAL development, and the integration of both layers. We spent 10 weeks in the APL development, 14 weeks in the VAL development, and 4 weeks with the integration of the layers. This case was our first VAS implementation and thus motivated an extra development and testing effort. It should also be noted that the AP04 provides more functionalities and complexity to take into account than the rest of tested autopilots.

The VAS adaptation for the Paparazzi autopilot took about 13 weeks: 8 weeks with the APL development, 3 weeks with the VAL development and 2 weeks with the layers' integration. The Paparazzi system includes its own simulation environment. Note that in this case practically all the VAL and the integration development was reused. This is why the VAL and the integration development effort have been reduced. On the other side, the APL code was

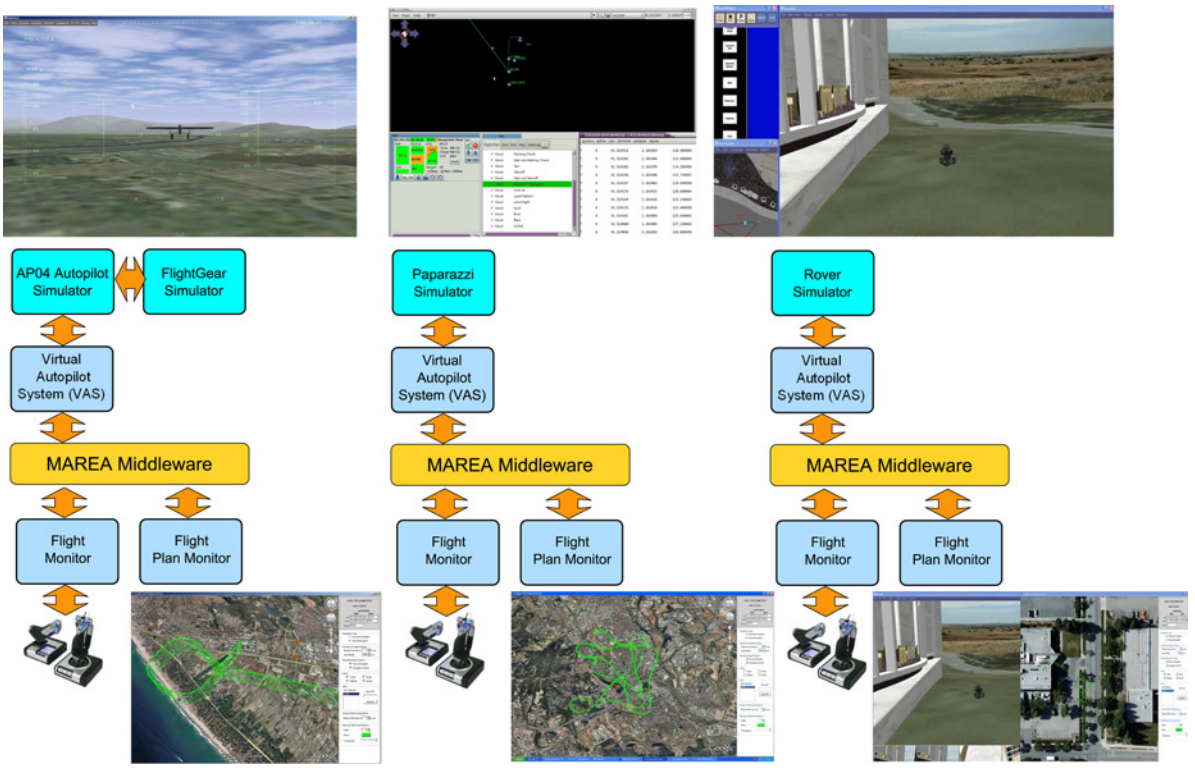

Fig. 14 FM, VAS, FPMo interacting with different autopilot simulators.

replaced in order to integrate the new autopilot to the USAL architecture. Additionally, the Paparazzi autopilot offers less complexity than the AP04 and therefore the learning curve is reduced.

In the UGV Reflection Rover autopilot case, we have taken about six weeks to carry out the autopilot integration into the USAL architecture. Again, the VAL was reusable from the AP04 and the Paparazzi versions but at the same time the lessons learned through the Paparazzi change allowed us to further reduce the VAL development and integration time. Most development effort was dedicated to implement the new APL for the Reflection autopilot, although small changes were required to adapt the PiC interfaces to a USAL version in which "altitude" was meaningless. In this last case and after full simulation and testing phases, VAS and USAL architecture were shortly tested on the real rover at the NASA Ames research center.

Working with different autopilots, we have been able to demonstrate how the VAS has overcome the clearly identified drawbacks of current UAS autopilot technology. In addition, we have been able to validate that the autopilot unit can be replaced by a different product and this change has almost no impact on the entire USAL except for the VAS. Thus, we have achieved and informally quantified development time reductions.

The concept of the combination of VAS + USAL has been employed to implement a helicopter-based UAS called Sky-Eye (see Fig. 15). This system is designed to improve the overall awareness of the fire managers by providing tactical support to wildfire monitoring and control of ground squads [31]. The Sky-Eye prototype is built around the AP04 autopilot and existing commercial off-the-shelf technology that can be immediately deployed on the field at a reasonable cost. Sky-Eye is designed to increase the level of UAS automation while being controlled from a mission point of view by a PiC. Information is gathered by the on board cameras, processed, and then relayed in such a way that it can be immediately exploited by the fire-fighter squads.

The Sky-Eye development has been greatly simplified thanks to the VAS. Initial prototypes were implemented by using the simulated version of the AP04. An almost immediate migration was possible to the real AP04, while a fixed wing aircraft version is currently under development using another commercial AP unit. The corresponding

Fig. 15 Sky-Eye UAS prototype based on the AP04 autopilot and the combination of VAS and USAL architecture.

interfaces will be implemented, and thanks to the VAS concept the overall mission-oriented architecture will be migrated from a tactical to a strategic monitoring platform with little effort.

VIII. Conclusion

This paper has presented the main aspects of the development of the VAS within the USAL architecture. We have shown that the VAS helps overcome some clearly identified drawbacks of current UAS autopilot technology. Based on the study of the state of the art in UAS autopilots, we identified that autopilots are highly oriented to flight capabilities and they do not take the overall UAS operation into account. The VAS proposes to use a well-defined range of states that are more focused on the full operation flow rather than just on the flight. Some states such as Takeoff and Landing make a clear step forward toward the integration of the UAS in a nonsegregated airspace from an operational point of view.

The VAS is designed following a Hardware Abstraction Layer concept; thus, the autopilot unit can be replaced by a new version or a different product, but this change will not have an unacceptable impact on the rest of UAS systems. In this way, we reduce the development effort when the autopilot is changed or updated by a newer version or the mission systems are deployed into a new UAS platform that uses a different autopilot unit. Telemetry is not autopilot dependent. The VAS offers the autopilot telemetry semantically grouped in order to be used by third party applications on board. The advantages identified during VAS development clearly highlight the urgent necessity for designing standards like the STANAG4586 [32], but well suited for civil use.

Finally, the VAS operates within the overall USAL architecture, surrounded by cooperating services that allow to implement much richer mission-oriented capabilities than those offered by current autopilots. The FPMa collaborates with the VAS to offer an almost unlimited number of waypoints, waypoint grouping, structured flight plan phases with mission-oriented legs with high-level semantics like repetitions, parameterized scans, etc; a contingency management strategy with built-in emergency alternatives; and a flexible concept of operation for airfield operations.

Future work needs to fully address the development of the safety aspects of VAS specification and implementation. Even though the VAS and the USAL add interesting features in terms of a well-structured contingency reaction strategy, they also add another layer of hardware and software that may fail. Tests beyond simulation of failure situations are necessary to fully evaluate the validity of the concept. The development of the conflict reaction capabilities with the VAS is also ongoing research. The VAS will support predefined sets of collision avoidance maneuvers that should be triggered by sense and avoid systems on board the UAS or commanded by the PiC. Finally, the exploitation of all these capabilities should enhance the level of automation of the overall UAS operation, providing powerful and usable HMTs to the PiC. The development of these interfaces is currently an ongoing effort out of the scope of this paper.

The applicability of the VAS concept has been tested working with three different autopilots (AP04, Piccolo, and Reflection). In all three cases, the development was successful and both simulation and small trials were performed. The full VAS + USAL architecture has been completed for its application in an helicopter-based platform, called Sky-Eye, devoted to monitor wildfires in tactical scenarios. The safe operation of this vehicle and the enormous amount of effort in developing the computational systems for this particular application is justified because we can guarantee that at any moment we will be able to migrate to another platform/autopilot retaining the development effort thanks to the VAS concept.

Acknowledgments

This work has been partially funded by Ministry of Science and Education of Spain under contract CICYT TIN2010-18989. This work has been also co-financed by the European Organization for the Safety of Air Navigation (EUROCONTROL) under its CARE INO III programme. The content of the work does not necessarily reflect the official position of EUROCONTROL on the matter.

References

[1] Iscold, P., Pereira, S., and Torres, A. B., "Development of a Hand-Launched Small UAV for Ground Reconnaissance," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 46, No. 1, Jan. 2010, pp. 335–348.

[2] Cox, T., Somers, I., and Fratello, D., "Earth Observations and the Role of UAVs: A Capabilities Assessment," Tech. Rep. NASA 20070022505, Aug. 2006.

[3] SC-203 RTCA, "Guidance Material and Considerations for Unmanned Aircraft Systems," Radio Technical Commission for Aeronautics, Document Do-304, Washington, DC, Mar. 2007.

[4] Tenoort, S., "Concept For Civil UAS Applications," Tech. Rep. D1.2, INOUI: Innovative Operational UAS Integration, May 2008.

[5] UAVNET, "European Civil Unmanned Air Vehicle Roadmap, Action Plan and Overview," Tech. Rep., 2005.

[6] Pastor, E., Lopez, J., and Royo, P., "UAV Payload and Mission Control Hardware/Software Architecture," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 22, No. 6, 2007, pp. 3–8.

[7] Royo, P., Lopez, J., Barrado, C., and Pastor, E., "Service Abstraction Layer for UAV Flexible Application Development," *Proceedings of the 46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA-2008-484, Vol. 13, No. 1, 2008, pp. 1–18.

[8] Ludington, B., Johnson, E., and Vachtsevanos, G., "Augmenting UAV autonomy," *IEEE Robotics and Automation Magazine*, Vol. 13, Aug. 2006, pp. 63–71.

[9] Meister, O., Frietsch, N., Ascher, C., and Trommer, G., "Adaptive Path Planning for VTOL-UAVs," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 24, No. 7, Aug. 2009, pp. 36–41.

[10] Gil, A., Passino, K., Ganapathy, S., and Sparks, A., "Cooperative Task Scheduling for Networked Uninhabited Air Vehicles," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 44, No. 2, July 2008, pp. 561–581.

[11] How, J., Fraser, C., Kulling, K., Bertuccelli, L., Toupet, O., Brunet, L., Bachrach, A., and Roy, N., "Increasing Autonomy of UAVs," *IEEE Robotics and Automation Magazine*, Vol. 16, No. 2, June 2009, pp. 43–51.

[12] Johnson, E., Proctor, A., Ha, J., and Tannenbaum, A., "Visual Search Automation for Unmanned Aerial Vehicles," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 41, No. 1, Jan. 2005, pp. 219–232.

[13] Sujit, P., and Ghose, D., "Search Using Multiple UAVs with Flight Time Constraints," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 40, No. 2, July 2004, pp. 491–509.

[14] Piccolo Family of Autopilots Page, http://www.cloudcaptech.com [retrieved 5 May 2011].

[15] UAV Navigation AP04 Product Page, http://www.uavnavigation.com [retrieved 5 May 2011].

[16] MP2028 Series Autopilots Page, http://www.micropilot.com/ [retrieved 5 May 2011].

[17] The Paparazzi Project, http://paparazzi.enac.fr/ [retrieved 5 May 2011].

[18] ArduPilot is a family of open source autopilots based on the Arduino platform, http://diydrones.com/ [retrieved 5 May 2011].

[19] Autonomous Vehicle Group at Aalborg University, http://www.es.aau.dk/projects/uav/ [retrieved 5 May 2011].

[20] Christophersen, H. B., Pickella, R., Neidhoefer, J. C., Koller, A. A., Kannan, S. K., and Johnson, E. N., "A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles," *Journal of Aerospace Computing, Information, and Communication*, Vol. 3, May 2006, pp. 187–213.

[21] Dong, M., Chen, B. M., Cai, G., and Peng, K., "Development of a Real-time Onboard and Ground Station Software System for a UAV Helicopter," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 8, Aug. 2007, pp. 933–955.
doi: 10.2514/1.26408

[22] Lopez, J., Royo, P., Pastor, E., Barrado, C., and Santamaria, E., "A Middleware Architecture for Unmanned Aircraft Avionics," *Proceedings of the ACM/IFIP/USEUNIX 8th International Middleware Conference*, Newport, CA, Nov. 2007.
[23] Santamaria, E., Royo, P., Barrado, C., Pastor, E., Lopez, J., and Prats, X., "Mission Aware Flight Planning for Unmanned Aerial Systems," *AIAA Guidance, Navigation and Control Conference and Exhibit*, AIAA, Honolulu, HI, Aug. 2008, pp. 1–21.
[24] Eurocontrol, "Guidance Material for the Design of Terminal Procedures for Area Navigation," Tech. Rep., May 2003.
[25] Pastor, E., Royo, P., Santamaria, E., Prats, X., and Barrado, C., "In-Flight Contingency Management for Unmanned Aerial Vehicles," *AIAA Unmanned...Unlimited Conference*, AIAA, Seattle, WA, April 2009, pp. 1–15.
[26] Delgado, L., Prats, X., Ferraz, C., Royo, P., and Pastor, E., "An Assessment for UAS Depart and Approach Operations," *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, AIAA-2009-6922, Sep. 2009, pp. 1–16.
[27] Pastor, E., Santamaria, E., Royo, P., López, J., and Barrado, C., "On the Design of a UAV Flight Plan Monitoring and Edition System," *Proceedings of the IEEE Aerospace Conference*, AIAA/IEEE, Big Sky, MT, March 2010.
[28] Ferraz, C., and Prats, X., "Design of Take-Off and Landing Operational Procedures for Unmanned Aerial Vehicles," Jul. 2009. http://upcommons.upc.edu/pfc/bitstream/2099.1/7288/1/memoria.pdf [retrieved 5 May 2011].
[29] Ippolito, C., "An Autonomous Autopilot Control System Design for Small Scale UAVs," Tech. Rep., QSS Group - NASA Ames Research Center, Dec. 2006, http://cmil.west.cmu.edu/publications.htm [retrieved 5 May 2011].
[30] Ippolito, C., Kaneshige, J., and Y, Y., "Neural Adaptive Flight Control Testing on an Unmanned Experimental Aerial Vehicle," *AIAA Infotech@Aerospace*, AIAA, May 2007.
[31] Pastor, E., Barrado, C., Royo, P., Santamaria, E., Lopez, J., and Salami, E., "Architecture for a Helicopter-Based Unmanned Aerial Systems Wildfire Surveillance System," *Geocarto International*, Vol. 26, Jan. 2011, pp. 1–19.
[32] NATO, "Standardization Agreement 4586 (STANAG 4586)," Tech. Rep., Nov. 2007.

Kelly Cohen
Associate Editor