Service Abstraction Layer for UAV Flexible Application Development

Pablo Royo, Juan López, Cristina Barrado and Enric Pastor*
Computer Architecture Dept., UPC, Spain

An Unmanned Aerial System (UAS) is an uninhabited airplane, piloted by embedded avionics and supervised by an operator on ground. Unmanned Aerial Systems were designed to operate in dangerous situations, like military missions. With the avionics technological evolution, Unmanned Aerial Systems also become a valid option for commercial applications, specially for dull and tedious surveillance applications. Cost considerations will also deviate some mission done today with conventional aircrafts to Unmanned Aerial Systems.

In order to build economically viable UAS solutions, the same platform should be able to implement a variety of missions with little reconfiguration time and overhead. This paper describes a software abstraction layer for a Unmanned Aerial System distributed architecture. The proposed abstraction layer allows the easy and fast design of missions and solves in a cost-effective way the reusability of the system.

The distributed architecture of the Unmanned Aerial System is service oriented. Functional units are implemented as independent services that interact each other using communication primitives in a network centric approach. The paper presents a set of predefined services useful for reconfigurable civil missions and the directives for their communication.

Nomenclature

UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
$USAL$	UAV Service Abstraction Layer
GPS	Global Positioning System
CPU	Central Processing Unit
SOA	Service Oriented Architecture
$UPnP$	Universal Plug and Play
VAS	Virtual Autopilot System
XML	Extensible Markup Language
$TCAS$	Traffic alert and Collision Avoidance System
$ADSB$	Automatic Dependent Surveillance-Broadcast
$FPMS$	Flight Plan Manager Service

I. Introduction

Even though the rapid evolution of UAS technology on airframes, autopilots, communications and payload, the generalized development of surveillance applications is still limited by the absence of systems that support the development of the actual UAV sensing mission. UAS engineers face the development of specific systems to control their desired flight-profile, sensor activation/configuration along the flight, data storage and eventually its transmission to the ground control. All these elements may delay and increase the risk and cost of the project. If realistic remote sensing applications should be developed, additional support to

*Computer Architecture Dept., Avda. del Canal Olimpic s/n, Castelldefels, Barcelona, 08860, Spain. Icarus group: http://icarus.upc.es/ {proyo, lopez, cristina, enric}@ac.upc.edu

effective system support must be created to offer flexible and adaptable platforms for any application that is susceptible to use them.

Focusing on civil applications, a wide range of application scenarios exists:[1,2] Remote environmental research, pollution assessment, fire-fighting management, security (i.e. border monitoring), agricultural and fishery applications, oceanography, communication relays for wide-band applications, etc. However no commercial solution exists nowadays that provides support for all these applications. The current challenge in UAS research is to define a flexible and reusable hardware/software architecture that permits the abstraction of the UAS functionalities for easy development of civil missions. The time to market is critical to achieve low cost solutions adequate for a competitive market.

In order to successfully accomplish this challenge, developers need to pay special attention to three different concepts: the flight-plan, the payload and the mission itself. The actual flight-plan of the UAS should be easy to define and flexible enough to adapt to the necessities of the mission. The payload of the UAS should be selected and controlled adequately. And finally, the mission should manage the different parts of the UAS application with few human interaction but with large information feedback. Many research projects are focused on only one particular application, with specific flight patterns and fixed payload. These systems present several limitations for their extension to new missions.

In this paper we propose a UAS Service Abstraction Layer (USAL)[3] as the shared common ground for all UAS applications. This is, a set of predefined services useful for reconfigurable civil missions. The list of services included in the USAL is completed with the description of their functionalities and the primitives for their communication. The proposed abstraction layer allows the easy and fast design of missions that solves in a cost-effective way the problem of reusability of the system. This service oriented architecture is managed by a middleware.[4,5] Mission functionalities are divided in simple isolated parts and implemented as independent services. These services interact each other using communication primitives in a network centric approach. They operate on a same level, on top of the middleware communication framework.

The paper is organized as follows: Section II shows an overview of the UAS service oriented architecture, the middleware and the communication primitives it offers to the services. In section III, the proposed *UAV Service Abstraction Layer* is presented as a solution to design UAV missions in a flexible and reusable way. Section IV.A presents the details of one important service of the USAL, the *Virtual Autopilot Service* (VAS) and its interface with the rest of the USAL. Section V details the operative modes of the VAS. Finally, Section VI concludes the article and shows the future lines of research.

II. System Overview

This section describes the architecture we propose for executing UAS civil missions: a distributed embedded system that will be on board the aircraft and that will operate as a payload/mission controller. Over the different distributed elements of the system we will deploy software components, called services, that will implement the required functionalities. These services cooperate for the accomplishment of the UAS mission. They relay on a middleware[5] that manages and communicates the services. The communication primitives provided by the middleware promote a publish/subscribe model for sending and receiving data, announcing events and executing commands among services.

II.A. Distributed Embedded Architecture

The proposed system is built as a set of embedded microprocessors, connected by a Local Area Network (LAN), in a purely distributed and scalable architecture. This approach is a simple scheme which offers a number of benefits in our application domain that motivates its selection:

- Development simplicity is the main advantage of this architecture. Inspired in the Internet applications and protocols, the computational requirements can be organized as services that are offered to all possible clients connected to the network.
- Extreme flexibility given by the high level of modularity of a LAN architecture. We are free to select the actual type of processor to be used in each LAN module. Different processors can be used according to functional requirements, and they can be scaled according to computational needs of the application. We denominate node to a LAN module with processing capabilities.

- Node interconnection is an additional extra benefit in contrast with the complex interconnection schemes needed by end-to-end parallel buses. While buses have to be carefully allocated to fit with the space and weight limitations in a mini/micro UAS, the addition of new nodes can be hot plugged to the LAN with little effort. The system can use wake-on-LAN capabilities to switched on a node when required, at specific points of the mission development.

II.B. Service Oriented Approach

Service Oriented Architecture (SOA) is getting common in several domains. For example, Web Services in the Internet world,[6] and Universal Plug and Play (UPnP)[7] in the home automation area. The main goal of SOA is to achieve loose coupling among interacting components. We name the distributed components as services. A service is a unit of work, implemented and offered by a service provider, to achieve desired final results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners.

The benefits of this architecture is the increment of interoperability, flexibility and extensibility of the designed system and of their individual services. In the implementation of a system, we want to be able to reuse existing services. SOA facilitates the services reuse, while trying to minimize their dependencies by using loosely coupled services. Loose coupling among interacting services is achieved employing two architectural constraints. First, services shall define a small set of simple and ubiquitous interfaces, available to all other participant services, with generic semantics encoded in them. Second, each interface shall send, on request, descriptive messages explaining its operation and its capabilities. These messages define the structure and semantics provided by the services. The SOA constraints are inspired significantly by object oriented programming, which strongly suggests that you should bind data and its processing together.

In a network centric architecture like SOA, when some service needs some external functionality, it asks the system for the required service. If the system knows of another service which has this capability, its reference is provided to the requester. Thus the former service can act as a client and consume that functionality using the common interface of the provider service. The result of a service interface invocation is usually the change of state for the consumer but it can also result on the change of state of the provider or of both services. The interface of a SOA service must be simple and clear enough to be easy to implement in different platforms, both hardware and software. The development of services and specially their communication requires a running base software layer known as middleware.

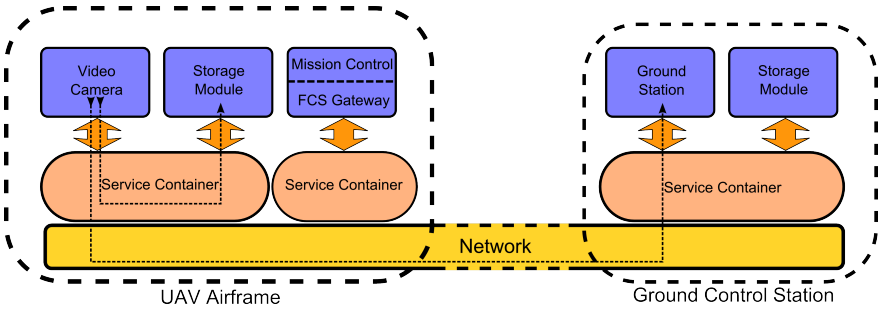

Figure 1. System Architecture.

II.C. Middleware

Middleware-based software systems consist of a network of cooperating components, in our case the services, which implement the business logic of the application. The middleware is an integrating software layer that provides an execution environment and implements common functionalities and communication channels. On top of the middleware, services are executing. Any service can be a publisher, subscriber, or both simultaneously. This publish-subscribe model eliminates complex network programming for distributed applications. The middleware offers the localization of the other services and manages their discovery. The middleware also handles all the transfer chores: message addressing, data marshaling and demarshalling (needed for services executing on different hardware platforms), data delivery, flow control, message retransmissions, etc.

The main functionalities of the middleware are:

- Service management: The middleware is responsible of starting and stopping all the services. It also monitors their correct operation and notifies the rest of system about changes in service behavior.
- Resource Management: The middleware also centralizes the management of the shared resources of each computational node such as memory, processors, input/output devices, etc.
- Name management: The services are addressed by name, and the middleware discovers the real location in the network of the named service. This feature abstracts the service programmer from knowing where the service resides.
- Communication management: The services do not access the network directly. All their communication is carried by the middleware. The middleware abstracts the network access, allowing the services to be deployed in different nodes.

Figure 1 shows the UAS distributed architecture proposed. Services, like the Video Camera or the Storage Module, are independent components executing on a same node located on the aircraft. Also on board, There is another node where the Mission Control service executes. Both nodes are boards plugged to the LAN of the aircraft. The mission has also some services executing on ground. The figure also shows two of them: the Ground Station service and a redundant Storage Service.

Each node of the UAS distributed architecture executes also a copy of the Service Container software. The set of all Service Containers compose the middleware which provides the four functionalities described above to the application services. This includes acting as the communication bridge between the aircraft and the ground. The middleware monitors the different communication links and chooses the best link to send information to ground or to air. From the service views the middleware builds a global LAN network that connects the LAN on ground and the LAN on board.

II.D. Communication Primitives

For the specific characteristics of a UAS mission, which may have lots of services interacting many-to-many, we propose four communication primitives based in the Data Distribution Services paradigm. It promotes a publish/subscribe model for sending and receiving data, events and commands among the services. Services that are producing valuable data publish that information while other services may subscribe them. The middleware takes care of delivering the information to all subscribers that declare an interest in that topic. Many frameworks have been already developed using this paradigm, each one contributing with new primitives for such open communication scenario. In our proposal we implement only a minimalistic distributed communication system in order to keep it simple and soft real-time compliant. Next, we describe the proposed communication primitives, which have been named as Variables, Events, Remote Invocations and File Transmissions.

Variables are the transmission of structured, and generally short, information from a service to one or more services of the distributed system. A Variable may be sent at regular intervals or each time a substantial change in its value occurs. The relative expiry of the Variable information allows to send it in a best-effort way. The system should be able to tolerate the lost of one or more of these data transmissions. The Variable communication primitive follows the publication subscription paradigm.

Events also follow the publication-subscription paradigm. The main difference in front of Variables is that Events guarantee the reception of the sent information to all the subscribed services. The utility of Events is to inform of punctual and important facts to all the services that care about. Some examples can be error alarms or warnings, indication of arrival at specific points of the mission, etc.

Remote Invocation is an intuitive way to model one-to-one of interactions between services. Some examples can be the activation and deactivation of actuators, or calling a service for some form of calculation. Thus, in addition to Variables and Events, the services can expose a set of functions that other services can invoke or call remotely.

In some cases, there also exists the need to transfer continuous media with safety. This continuous media includes generated photography images, configuration files or services program code to be uploaded to the middleware. The File Transmission primitive is used basically to transfer long file-structured information from a node to another.

III. USAL: UAS Services Abstraction Layer

Providing a common infrastructure for communicating isolated avionics services is not enough for keeping the development and maintenance costs for UAV systems low. The existence of an open-architecture avionics package specifically designed for UAS may alleviate the developments costs by reducing them to a simple parameterization. The design of this open-architecture avionics system starts with the definition of its requirements. These requirements are defined by the type of UAS (mini or tactical UAS in our case) and the mission objectives. From the study and definition of several UAS missions, one can identify the most common requirements and functionalities that are present among them.[1,2] Once the most common requirements have been identified how we have organized and implemented them, creating an abstraction layer called UAS Service Abstraction Layer (USAL).

The final goal of the USAL is twofold:

1. First, reduce time to market when creating a new UAS system. The USAL together with middleware will simplify the integration of all basic subsystems (autopilot, communications, sensors, etc) because it will already provide all required glue logic between them.

2. Second, simplifying the development of all systems required to develop the actual mission assigned to the UAS. In most cases reducing this complexity to specifying the desired flight plan and sensor operation to some of the services available in the USAL and parameterizing the specific services to be used every time.

Using the USAL allows to abstract the UAS integrator from time consuming and complex underlying implementation details. The USAL and middleware offer a light weight service-based architecture with a built-in inter-service communication infrastructure. A large number of available services can be selected to create specific configurations, while new services can be easily created by inheriting exiting ones; e.g. to integrate a new camera with some specialized behavior.

Even though the USAL is composed of a large set of available services, not all of them have to be present in every UAS or in any mission. Only those services required for a given configuration/mission should be present and/or activated in the UAS. Available services have been classified in four categories:

1. Flight services: all services in charge of basic UAS flight operations: autopilot, basic monitoring, contingency management, etc.

2. Mission services: all services in charge of developing the actual UAS mission.

3. Payload services: specialized services interfacing with the input/output capabilities provided by the actual payload carried by the UAS.

4. Awareness services: all services in charge of the safe operation of the UAS with respect terrain avoidance and integration with shared airspace.

Figure 2 provides a global vision of most services that can take part in a UAS mission, most of them already predefined in the USAL.

The rest of this section will describe the different service groups that conform the USAL. For each group their responsibilities and global behavior are detailed. In addition, their service components and care shown.

III.A. Flight Services

Nowadays many autopilot manufacturers are available in the commercial market for mini/micro UAS.[8] 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 UAS is a complex task because none of them is mutually compatible. Moving from one autopilot to another may imply redesigning from scratch all the remaining avionics in the UAS. Therefore, once an autopilot is selected it may remain in the system for all its operational live.

Current UAS autopilots also have two clearly identified drawbacks that limit their effective integration with the mission and payload control inside the UAS:

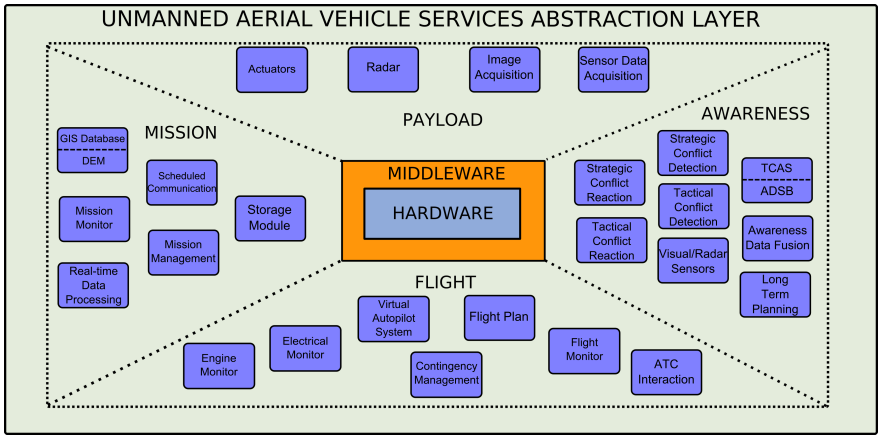

Figure 2. Overview of the USAL service-based architecture.

1. The complexity of exploiting on-board the autopilot telemetry by other applications in the UAS is extremely complex and autopilot dependent. Autopilots telemetry is typically designed just to keep the UAS state and position under control and not to be used by third party applications.

2. The flight plan definition available in most autopilots is just a collection of waypoints statically defined or hand-manipulated by the UASs operator. However, no possible interaction exists between the flight-plan and the actual mission and payload operated by the UAS.

The flight services are a set of standardized architecture components designed to operate as interface between the real autopilot and the rest of subsystems in the UAS; e.g. the mission and payload services. The objective of the flight services is multiple:[9]

- Abstract autopilot details and peculiarities to the rest of the system.
- Extract internal sensor information from the autopilot and offer it to other services for its exploitation during the UAS mission.
- Provide a common flight-plan definition, improving by large actual commercial autopilot capabilities by adding a flight management layer on top of them.
- Provide status monitoring capabilities and automatic contingency management for efficient emergency response.

Figure 3 shows the fundamental components in the flight services category as well as the major relations among them.

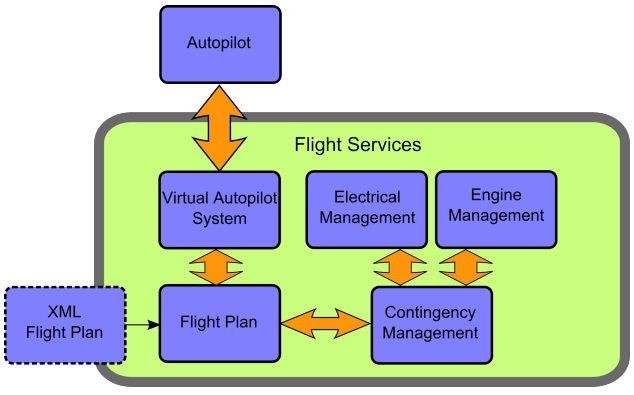

Figure 3. Overview of the available flight service category.

The *virtual autopilot system* (VAS) is a service that on one side interacts with the selected autopilot and therefore needs to be adapted to its peculiarities. The VAS operates similarly as drivers work on operating systems abstracting away the implementation details from actual autopilot users. For other services in the UAS, 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 the VAS 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. However, the way this information is provided to its clients (event-based, continuous, and actual transmission rate) can be selected according to each individual client thanks to the communication infrastructure offered by the service-based architecture. In order to well understanding the USAL services; the VAS is shown in detail in Section IV.A.

The flight planning capabilities of all autopilots are dissimilar, but generally limited to simple waypoint navigation. In some cases they include automatic take-off and landing modes. From the point of view of the actual missions or applications being developed by the UAS using a simple waypoint-based flight plan may be extremely restrictive. Above the VAS we develop a *flight plan manager*. It has been designed to implement much richer flight-plan capabilities on top of the available capabilities offered by the actual autopilot. The flight plan manager offers an almost unlimited number of waypoints, waypoint grouping, structured flight-plan phases with built-in emergency alternatives, mission oriented legs with a high semantic level like repetitions, parameterized scans, etc.

Actual flight plans can be introduced through an RNAV-inspired[10] XML formalism. Additionally, all available highly semantic legs can be modified by other services in the UAS by changing a number of configuration parameters without having to redesign the actual flight-plan; thus truly allowing cooperation between the autopilot operation and the specific UAS mission.

Given that, in general, the real autopilot capabilities will be much simpler that those available in the flight plan manager, additional waypoints will be generated according to requirements. These internal waypoints will be dynamically feed into the autopilot through the VAS during the mission time, therefore transforming the flight manager in a virtual machine capable of executing flight plans. As a result, combining both the abstraction mechanism provided by the autopilot abstraction service and the increased flight plan capabilities of the flight-plan manager, we obtain a highly capable platform that can be easily integrated to perform much more efficiently a number of valuable missions.

The *engine and the electrical managers* are in charge of monitoring the engine and electrical parameters of the UAS in order to detect problems and notify a contingency management. Furthermore of monitoring functions, both services can estimated the rest of time which the mission can be developed in perfect conditions. The engine management service controls all the parameters involved in the engine, such as temperature, oil, fuel, etc. The service manages the correct range of each parameter; these ranges can be configured for each engine. When the engine manager detects whatever parameter out of range, an event is raised in order to inform the contingency manager service. The electrical manager service monitors the electrical consumption of all the devices (hardware) and as the engine manager inform the contingency manager when the UAS has low power.

The *contingency manager* is responsible to collect or status information related to multiple sources: engine, electrical, fuel, communications, etc. identifying if some type of contingency has evolved and deciding which type of reaction is required. The reaction will be more drastic depends on the sort of contingency. It means that some situations will be solved for itself instead of inform the Flight plan manager or other reaction services. If the flight plan manager has to solve the contingency, it should be informed about the decision so that it can react in accordance to the predefined set of alternative landing strips or executing emergency landing procedures.

III.B. Mission Services

After many years of development, UAS are reaching the critical point in which they could be applied in a civil/commercial scenario. However, we believe that payload and mission control are the main bottlenecks (together with the lack of UAS legislation) that may prevent such development:

- Too much human control from a ground station is still required. Flight control computers do not provide additional support beyond basic flight plan definition and operation. Additionally, payload should be also remotely operated with very little automation support.

- Economical efficiency requires the same UAS to be able to operate in different application domains. This necessity translates into stronger requirements onto the mission/payload management subsystems, with increased levels of flexibility and automation.

Figure 4 shows the fundamental components in the mission and payload services category as well as the major relations among them. A highly automated mission operation is necessary in order to increase system productivity and reduce the costly and error prone manual control from ground stations.

The *mission manager* is the orchestra director of the USAL services. This service supervises the flight services and the payload services; as well as the coordination of the overall operation. The mission manager executes a user defined automata with attached actions (i.e. service activations) at each state or transition. Actions can be predefined built-in operations or specific pieces of user code. In particular the mission manager is capable of modifying the actual flight plan by redefining its parameters or by defining new stages or legs.

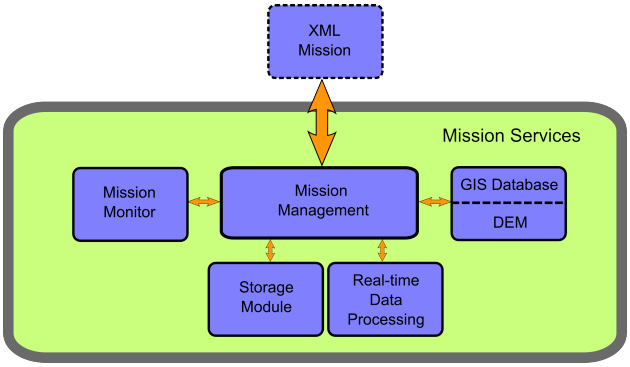

Figure 4. Composition of mission service category.

The *real-time data processing* service gives the intelligence for complex missions. This service offers predefined image processing operations (accelerated by FPGA hardware if available) that should allow the mission manager to take dynamic decisions according to the actual acquired information.

The *storage service* saves video streams, photos and telemetry to subsequently process them on the ground. In many cases there is not enough communication bandwidth to the ground, and therefore only vital data could be sent, while the rest should be stored in an orderly fashion such that it can be properly exploited. The storage service permits to save system configuration information, just in case if any service needs to re-start or re-install on board. This service not only works as data base; it operates as file system in an operating system; offering function as searching data, creates new memory spaces for a new services, backup functions, etc.

The *digital elevation model* (DEM) service is involved in both flight and mission management of the UAS. If it is necessary flight at low altitude the autopilot needs a good DEM to know how is the terrain orography. The DEM service allows to detect altitude contingencies. These contingencies will be managed by the awareness group of services. .

The *geographic information service* (GIS) can generate orography information too. However the GIS not only has altitude information; it has all sort of geographic information: *e.g.* cities, villages, mountains, big forest, etc. This information will be useful in order to decide flight plan changes. However these systems are rather heavy computationally. In this case, the GIS will be placed on ground station, and only the important geographic information will be sent to an equivalent service in the UAS.

III.C. Awareness Services

A UAV System is a highly instrumented aircraft and has no pilot on board. With these conditionings the more suitable flight rules for a UAV are IFR, however for remote sensing missions the advantages of UAV systems is precisely its capacity for flighting at any altitude, where VFR aircrafts are found. UAVs must relay on its instrumentation equipment to properly inform the pilot in command on the ground or substitute the pilot capacities in VFR conditions. The awareness services are responsible of such functionalities. Flight Services are in charge of the aircraft management in normal conditions while the Awareness Services are in charge of monitoring surroundings conditions and overtake aircraft management in critical conditions. In

this case mission services come to a second priority, until flight conditions become again normal. The lists of flight services are (see Figure 5):

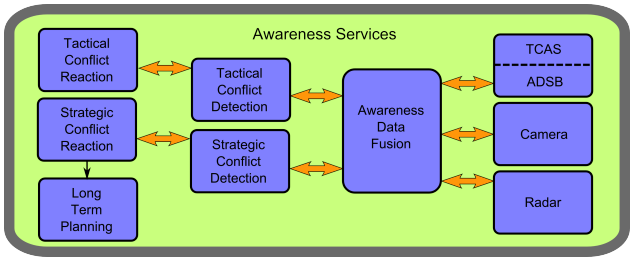

Figure 5. Composition of awareness service category.

The *data fusion service* stores the current air traffic, meteorological and terrain state of the aircraft as seen by the different sensors of the aircraft. There are many devices used to acquire awareness information, for example: TCAS, ADSB, cameras, meteorological sensors, radar, DEM, etc. We need a good matching of all of this information in order to offer a correct and quick reaction. This service fusions all data received in a centralized global view. All of this data is sent to the conflict detection service in order to manage possible tactical and strategic conflicts.

The *TCAS and ADSB services* monitor the airspace around an aircraft, independent of air traffic control, and warns the UAS of the presence of other aircraft which may present a threat of mid air collision. These services act as a sensor in the awareness services group, their information is sent to the data fusion service to offer a global view and choose the correct decision anytime.

The *radar service* uses electromagnetic waves to identify the range, altitude, direction, or speed of both moving and fixed objects. In our case its use are especially to identify other aircrafts and weather formations. As TCAS or ADSB services the Radar service will send information to the data fusion service in order to evaluate it.

The *conflict detection service* studies what is happening around the UAS and generates alert when detect something that could be dangerous. Furthermore, this service is in charge of classify how danger is the conflict detected. First, the conflict detection evaluates the hazard, then the service decides if the conflict needs a tactical reaction or strategic reaction in order to solve it.

The *tactical reaction service* generates reactions in case of a quick response is needed. This reaction will have the maximum priority level in the whole system. This service will take virtual autopilot system control, and it will manage the UAS flight canceling the flight plan manager commands and solving the tactical conflict.

The *strategic reaction service* generates reactions in the case of strategic conflicts. This conflict are detected with enough time in order to plan a strategy and solve the problem. The reaction may involve interaction with the flight plan manager or the mission management.

The *long term planner service* takes part in flight just generating complex trajectories. In this way the functionalities of flying the plane and calculating the complex flight paths are separated in different services. The strategic reaction service delegates to the Long Term Planner Service the generation of a complex path in order to solve conflicts.

III.D. Payload Services

Payload services are defined for low level devices, mainly raw data acquisition sensors that need to be processed before being used in real-time or stored for post-mission analysis. The complete list of services is directly related to available sensors, and except for most classical cameras they need to be created or adapted by the end user. However, USAL offers pre-build skeletons that should be easily adapted for most common devices.

IV. USAL Use Case: Virtual Autopilot Service (VAS)

Now that the USAL and their functional service groups have been shown, we are going to describe in detail one of the most important services, the Virtual Autopilot Service. This section will deeply explain

this service at both the functional and architectonic levels. After that, the data communications that this service generates or receives will be shown. This communications are grouped in three different groups: flight monitoring, navigation information and status & alarms.

Flight monitoring comprises all the output communications that the VAS provides for abstracting the telemetry capabilities of the real autopilot. Navigation mainly describe the interrelationship with the services that store and manage the flight plan of the mission. Finally, the status & alarms notifies of problems in the VAS both at the software and hardware levels.

IV.A. VAS Functional Overview

The Virtual Autopilot Service is the component that will interact between the autopilot and the rest of the components of the USAL. After studying several UAS autopilots we have seen that their functioning and capabilities are very similar, however their implementation details greatly differ.[8] To improve the flexibility of a UAS is needed to abstract the concrete autopilot used. The main objective of the VAS is to implement this abstraction layer, to isolate the system of autopilot hardware changes. As every UAS mission needs to control autonomously the aircraft following a list of waypoints, the VAS clearly belongs to the minimum services required in the USAL.

The VAS is specially suited to work in conjunction with the Flight Plan Manager Service (FPMS). This service the stores the flight plan of the mission which is composed of a complex hierarchy of waypoints that the UAS must fly. The VAS will command the hardware autopilot to guide the UAS flight and it will ask the FPMS for new waypoints as the autopilot is consuming them.

VAS operates similarly as drivers work on operating systems, abstracting away the implementation details from actual autopilot users. This service may also offer all required flow of data to any other application on board the UAS.

The flight planning capabilities of all autopilots are dissimilar, but generally limited to simple waypoint navigation and in some cases they include automatic take-off and landing modes. From the point of view of the actual missions or applications being developed by the UAS using a simple waypoint-based flight plan may be too restrictive. In addition to the VAS a Flight Plan Manager Service (FPMS) will be designed to implement much richer flight-plan capabilities than offered by the actual autopilot. The FPMS offers structured flight plan phases with built-in emergency alternatives, leg based path description, mission oriented legs with a high semantic level like repetitions, parameterized scans, etc. Figure 6 provides an schematic view of the relation between VAS and FPMS inside the USAL.

Given that the real autopilot capabilities will be much simpler than those available in the FPMS, additional waypoints will be generated according to requirements. Internal waypoints will be dynamically fed into the autopilot through the VAS during the mission time, therefore transforming the FPMS in a virtual machine capable of *executing* flight plans. As a result, combining both the abstraction mechanism provided by the VAS and the increased flight plan capabilities of the FPMS, we obtain a highly capable platform that can be easily integrated to perform much more efficiently a number of valuable missions.

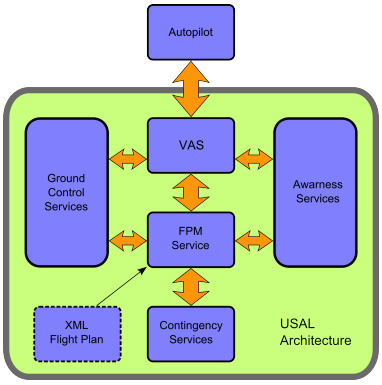

Figure 6. Architecture of the VAS FPM Service inside the USAL.

IV.B. VAS Architecture

As seen in section IV.A, VAS is a service that provides a standardized interface to the particular autopilot on board. Since it directly interacts with the selected autopilot it needs to be adapted to its peculiarities. For other services in the UAS, 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, 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. VAS belongs to the set of services defined in the UAS Service Abstraction Layer (USAL) and will provide flight monitoring and control capabilities to other services.

The inclusion of the VAS greatly improves the flexibility of the system. 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. Another important motivation is to provide an increased level of functionality. VAS should permit operation with a virtually infinite number of waypoints, thus overcoming a limitation present in all studied UAS autopilots. It will also be able to check the plausibility of these waypoints. This increased level of functionality includes the capability to take control of the flight and generate new waypoints in response to contingencies when services in charge of navigation control fail.

UAS autopilots available today are similar in their operation and capabilities, though their implementation details greatly differ. The key to carry out a correct abstraction is to offer in the VAS interface the common functionality and data that can be found in any autopilot. This information will be organized in the following four groups:

- flight monitoring information (also referred to as telemetry),
- navigation information,
- status/alarm information, and
- flight state management.

The first group relates to the need of the autopilot to acquire and process attitude and position data. The second one is needed to determine the path that the aircraft will follow. The third, gives information about its current status and possible alarms. Finally, the last one is added to the VAS design to provide the aforementioned increased level of functionality. This last group will change the autopilot states when necessary. Figure 7 shows the different parts of the VAS service. As displayed in the figure, monitoring and status/alarms information are outgoing flows, while navigation and state management have input/output direction.

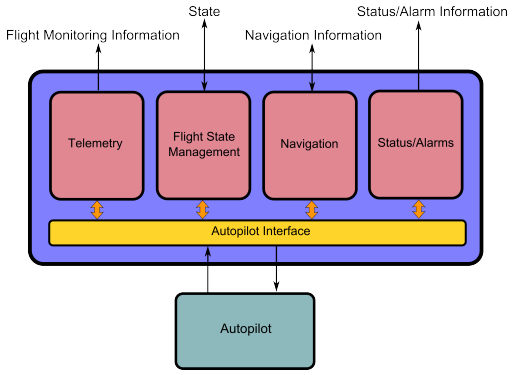

Figure 7. Information flow inside the Virtual Autopilot System.

This four groups of interfaces are described more in detail in the following subsections.

IV.C. Flight Monitoring Services

There are several ways to expose the information that generate the sensors of the autopilot. Usually, the autopilot manufacturer groups all this information in large packets of data, that are sent via a radio modem

at a certain frequency. In our service-based architecture, the VAS service will offer this information over a LAN to all the services that need this information. The information will be semantically grouped in a way that this information relates to parameters, situations or attitudes of the aircraft, independently of the real autopilot hardware and sensors. Table 1 shows all the Flight Management Information which the VAS will publish over the rest of the services on the network.

Table 1. Flight Management Information published by VAS.

Protocol Primitives	Name	Composition	Data Type	Range	Unit	Description
Variable	uavAngles	Roll Pitch Yaw	Float	0 to 2π rad.	radians	Roll, Pitch and Yaw angles of the UAS
Variable	uavAcceleration	X Y Z	Float	Device Range	m/s^2	Acceleration in UAS X, Y and Z axis.
Variable	uavRateTurn	X Y Z	Float	UAV Range	rad/s	Rate of turn in UAS X, Y and Z axis
Variable	uavPosition	Latitude Longitude Altitude(MLS) Pressure Altitude	Double Double Float Float	0 to 2π 0 to 2π UAV Range	radians radians meters meters	3D UAV Position
Variable	uavSpeed	North East Down	Float Float Float	UAV Range	m/s	3D speed in the UAV
Variable	uavAirspeed	Indicated airspeed True airspeed	Float Float	UAV Range	m/s	Air speed data in UAV
Variable	windEstimated	North East Down	Float	UAV Range	m/s	North, east and down wind speed estimated
Variable	missionTime	Time	Integer	UAV Range	ms	Mission Duration

UAV Angles

Indicates the instantaneous angular position of the system. Roll, pitch and yaw are offered in radians.

UAV Acceleration

Indicates the instantaneous acceleration vector of the system. X, Y, Z vector is offered in meters per square second.

UAV Rate of Turn

Indicates the instantaneous rate of turn of the system. X, Y, Z turn vector is offered in radians per second.

UAV Position

Indicates the instantaneous position of the system. Latitude, longitude, altitude (MSL) and barometric pressure altitude are offered in radians and meters respectively.

UAV Speed

Indicates the instantaneous speed vector of the system. North, east and down speed vector is offered in meters per second.

UAV Air Speed

Indicates the instantaneous speed vector of the system relative to the air. North, east and down speed vector is offered in meters per second.

Wind Estimate

Indicates a wind direction estimation around the system. North, east and down wind vector is offered in meters per second.

Mission Time

Indicates the actual mission time in seconds since the VAS start-up or the last mission time reset.

In general, UAS autopilots provide lots of information from all the sensors; however only a little part of them is really useful to implement a system to develop missions. Flight Monitoring Information has been chosen thinking in what information could be useful for the mission development.

IV.D. Navigation Services

In the USAL architecture, the Flight Plan Manager Service is in charge of generating the navigation commands to the VAS. In most cases these commands will take the form of waypoints or requests for changing the autopilot state. The VAS feeds the autopilot with its internals waypoints as it consumes system waypoints and commands.

IV.D.1. Output Navigation Services

The next group of information is the output Navigation group. This information basically states where to the UAV is going at any moment, in which direction is moving and which waypoint is flying. Table IV.D.1 defines this information.

Table 2. Navigation Information provided by VAS.

Protocol Primitives	Name	Composition	Data Type	Range	Unit	Description
Event Function	currentWp	Latitude Longitude Altitude(MLS) Number	Double Double Float Integer	0 to 2π rad. 0 to 2π rad. UAV Range No Range	radians radians meters	Read waypoint information where the UAS goes.
Function	previousWp	Latitude Longitude Altitude(MLS) Number	Double Double Float Integer	0 to 2π rad. 0 to 2π rad. UAV Range none	radians radians meters	Read the previous waypoint information.
Function	rwySituation	Latitude Longitude Altitude(MLS) Heading Length	Double Double Float Float Integer	0 to 2π rad. 0 to 2π rad. UAV Range 0 to 2π rad. None	radians radians meters radians meters	Read the runway information from the autopilot
Function	altRwySituation	Latitude Longitude Altitude(MLS) Heading Length	Double Double Float Float Integer	0 to 2π rad. 0 to 2π rad. UAV Range 0 to 2π rad. None	radians radians meters radians meters	Read the VAS alternative runway information
Event	uavDirection	Indicated airspeed Altitude(MLS)	Float Float	UAV Range UAV Range	m/s meters	Current target airspeed, altitude and heading.
Event Function	vasState	State	Integer	(See (section V)	N/A	State that VAS supports

Current Waypoint

Indicates the current waypoint where the system is flying. This information is offered as an event every time the autopilot switches from one waypoint to the next. Also offered as a function. Waypoints are indicated using the USAL waypoint specification format.

Previous Waypoint

Indicates the previous waypoint where the system was flying to. This information only offered as a function and it is intended to compute the previous leg that the autopilot was flying. Waypoints are indicated using the USAL waypoint specification format.

Runway Situation

Indicates the position of the selected runway for normal operations. Runways are defined as runway entry points, runway direction and runway lengths. This information only offered as a function and it is intended to confirm the data stored in the VAS.

Alternative Runway Situation

Indicates the position of the selected runway for alternative operations. This runway is intended to be used in case of emergency if the UAV cannot reach the prefered runway (not enough fuel, batery,

structural damage, etc.). Runways are defined as runway entry points, runway direction and runway lenghts. This information only offered as a function and it is intended to confirm the data stored in the VAS.

UAV Direction

Indicates selected direction of the UAV to flight. This direction will depend on the selected autopilot mode: waypoint navigation, directed mode, hold mode, etc. It identifies its target bearing, airspeed and altitude.

VAS State

Indicates the actual state of the VAS system. The supported states and a brief description is included in Table 3. State is reported each time the VAS switches from one state to another; and each time an state change is requested but cannot be fulfilled.

All of this service information will be mainly used for the FPMS in order to interact dynamically with VAS and the mission services. In case the FPMS or some other service implied in the UAS flight fails, the VAS can take control of the UAS. This emergency state implies a change in the flight state management to Safe mode until the flight plan is recovered or the UAS lands safely. To support the Safe mode the VAS incorporates extra functionalities. One of them is the ability to change the main runway to a closer one. Therefore, the VAS stores alternative runways just in case a contingency situation happens and an emergency landing is needed.

IV.D.2. Input Navigation Services

The next group of information is the input Navigation group. This information basically tells the VAS configuration parameters for the autopilot operation, as well as parameters to configure the operative parameters of the states in which the VAS may operate. The next table defines this information.

QNH Ground Pressure

Sets the QNH pressure value at the main runway for the pressure altimeter.

Ground Level Altitude

Set the ground level altitude to the autopilot. The VAS does not have a digital elevation model, so this is the only way for the system to know the ground level.

Max Mission Time

Once the VAS system starts up a Mission Time timer starts. The Max Mission Time sets up the limit operation time. If this limit is surpassed an alarm will be raised.

Deflection Surface Control

Set the ground level altitude to the autopilot. The VAS does not have a digital elevation model, so this is the only way for the system to know the ground level.

New Waypoint

The system uses this packet to feed up the autopilot with the mission waypoints. With this packet we will compose the flight plan of the mission, that is, the points over the UAS will fly.

Uav Speed

Set target indicated airspeed. This packet only have effect once in directed state. However, each waypoint defines the speed which the UAS will have to arrived over that waypoint.

Uav Altitude

Set target altitude. As uav speed, this packet only have effect once in directed state and each waipoint defines the altitude which the UAS will have to arrived over that waypoint.

Table 3. Input Navigation Information by VAS.

Protocol Primitives	Name	Composition	Data Type	Unit	Description
Event	qnhGround	Pressure	Float	Pascals	Sets the QNH pressure for the pressure altimeter
Event	gndLevel	Ground level altitude(MLS)	Float	Meters	Set the ground level altitude to the autopilot
Event	maxTimeMission	Time	Integer	ms	Set the mission time.
Event	deflecSurfCont	Aileron servo deflection Elevator servo deflection Ruder servo deflection Throttle servo deflection	Short Integer Short Integer Short Integer Short Integer	Radians Radians Radians Radians	Control packet for direct action on Surface
Event	newWp	Latitude Longitude Altitude(MLS) Number Speed	Double Double Float Integer Float	Radians Radians Meters m/s	Set coordinates of flight plan waypoint.
Event	uavSpeed	IAS	Float	m/s	Indicated air speed of UAV
Event	uavAltitude	Altitude(MLS)	Float	Meters	UAV altitude
Function	newMainRwy	Latitude Longitude Altitude(MLS) Heading Length	Double Double Float Float Integer	Radians Radians Meters Radians Meters	Set the coordinates of the main runway.
Function	newAltRwy	Latitude Longitude Altitude(MLS) Heading Length	Double Double Float Float Integer	Radians Radians Meters Radians Meters	Set the coordinates of the alternative runway.
Event	changeVasState	State	Integer	N/A	Set the VAS state.
Event	clearWps	Event	Integer	N/A	Clear all the flight plan waypoints
Event	skipLeg	Leg identifier	Integer	N/A	Skip one leg and pass to another.
Event	discardLegWp	Waypoints identifier	Integer	N/A	Discard a range of waypoints of the flight plan.

New Main Runway

Set coordinates of runway. The main runway is where the UAS will land. In principle the main runway is where the UAS will take off. However, the UAS will sometimes need a closer runway to landing, specially if it have happened any contingency and the UAS need to land in order to solve the problem.

New Alternative Runway

Set coordinates of alternative runway. In the alternative runway, the UAS will save alternative places where the UAS can land. The purpose of this packet is to store a closer runway where the UAS is flying each moment.

Change VAS Mode

Sets the current VAS state. With this packet the some services as FPMS, ground station service or see and avoid service can switch the VAS states. The VAS states are described more deeply in section V.

Clear Waypoints

This packet is used to clear all the flight plan waypoints. Sometimes, we may want to change the mission flight plan for another one. First of all, we have to cancel the actual flight plan with this packet and then we can upload new one.

Skip Leg

Legs specify the path that the plane must follow in order to reach a destination waypoint from the previous one. In some cases we may need to skip a whole leg instead of all the waypoints. This packet permits skip one leg and pass to another.

Discard Leg Waypoint

The FPMS controls all the mission flight plan. In some cases, the FPMS may be interesting in discard a range of waypoints of the flight plan. These waypoints can take several legs or can be part of a leg.

IV.E. VAS and Autopilot Status/Alarms

The VAS is the interface between the autopilot and the rest of the system. So is in charge of inform the status of the autopilot and its own status. A autopilot is a complex hardware that needs to be monitoring every time. With this group of packets we can monitor the autopilot and the VAS status; when any part of these device has a failure the VAS will send an alarm to the network. All of these alarms are sending by the network as events for two reasons. First, because the alarms are very important for the system and it is needed that these notifications safely arrive to all the services that process them. Second, we will only need to know the status when something is wrong. They are not a periodical information like the telemetry flows. Next table shows different alarms of the VAS:

Table 4. Status and Alarms Information.

Protocol Primitives	Name	Description
Event	groundComLossAlarm	Ground Communication loss.
Event	gpsApAlarm	GPS Alarm.
Event	outRangeTempApAlarm	Temperature outside range.
Event	voltSysApAlarm	Bus voltage alarm (System).
Event	voltServApAlarm	Bus voltage alarm (Servos).
Event	accApAlarm	Autopilot acceleration alarm.
Event	rateTurnApAlarm	Autopilot rate of turn alarm.
Event	ImuApAlarm	Autopilot IMU alarm.
Event	magnetometerApAlarm	Autopilot magnetometer alarm.
Event	pressureAltimeterApAlarm	Autopilot pressure altimeter alarm.
Event	anemometerApAlarm	Autopilot anemometer alarm.
Event	missionAlarm	Mission time reached alarm.
Event	wpRangeAlarm	Waypoint outside range alarm.
Event	wpProcessAlarm	Error processing parameters.
Event	lackMainRwy	None main runway.
Event	speedAlarm	Speed outside range.

The most important alarms are the ones that monitor the status of the link between the UAV and the ground control station (Ground Communication Loss Alarm) and the links between the GPS receiver and the satellites (GPS Autopilot Alarm). Other important situation that should be controlled is the voltage of the different hardware components of the UAS. Low voltages clearly indicates problems in power system (batteries, generators or the signal conditioning system) and usually will end in a global malfunctioning of the system.

The VAS also provides detailed alarms for notifying of problems in the different sensors inside the autopilot (Accelerometer, Gyroscope, Magnetometer, Anemometer and Pressure altimeter). UAS operation can usually continue however on a degraded mode. In this case, the UAS will usually try to return to base for maintenance.

Finally, some alarms manage the specific operation of the VAS service. The Waypoint Range alarm will be raised when the Flight Plan service (or the service on charge of providing the waypoint list to the autopilot) tries to feed a waypoint that it is not coherent with the rest of the flight plan (too far away from the previous waypoint). This usually indicates a problem with the flight plan service or the flight plan itself.

Other specific situation that it is notified by the UAS, it is the case when a main runway has not been programmed. This runway is needed for some calculations and it is also the place where the UAS will try to carry an emergency landing. Finally, in case of an internal error the VAS will raise the Process Error alarm, indicating that it has detected some anormal behaviour in his internal calculations.

V. VAS Operational States

During its usual operation an autopilot have different forms of behavior: take-off, waypoint navigation, directed flight, landing, etc. Depending on the implementation and the capabilities of the concrete autopilot, these behaviors or states will differ. Some sort of standardization and adaptation is needed if the VAS has to make interoperable different autopilots.

In this section we are going to describe a general overview of the VAS operational states. For lack of space, the detailed the description of the states and the transitions are omitted. The commercial autopilots are very focused in flight states, however a mission can be composed of many different states, for example, we can have different behaviors for the contingencies, which need different types of response. Many autopilots solve this sort of problems just coming back to base station. However, we want the UAS to be able to enter in safe states where it can try to recover the situation.

Obviously, not all the commercial autopilots will implement the full range of states provided by the VAS and it is responsibility of each VAS implementation to fulfill the whole functionalities needed. For example, in case that a concrete autopilot does not have take-off mode, its companion VAS will have to implement a take-off algorithm.

The figure 8 shows all the VAS states. This is a graph diagram in which each node corresponds to one state. In each state, the UAS develops several tasks in order to achieve the goals of the mission in each moment. All the related states are grouped together. The initial state inside each group is showed with an arrow on the top right box state corner. The rest of the arrows show the transitions between the different states. The diagram is descendant from the beginning of the mission to the end, although, in some case, there are horizontal transitions.

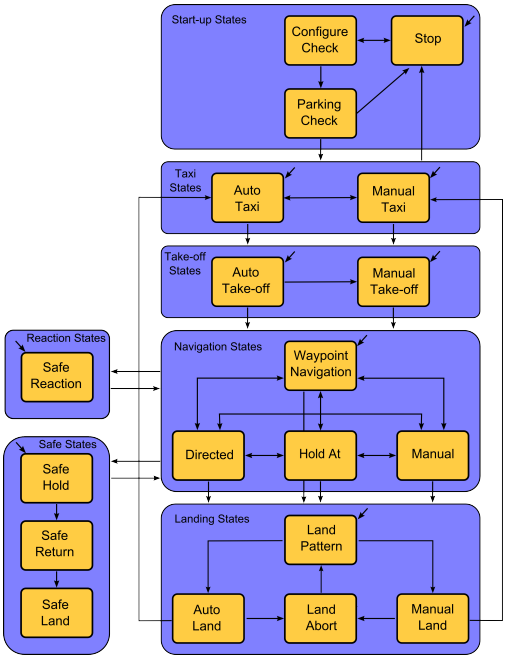

Figure 8. VAS Operational States ant Transitions.

Start-Up States

The first states related to system initialization are arranged in the Start-Up group. The initial state when the system is switched on is the Stop state. In this state, the UAS is stopped with all the devices of the system on. The next state is the Configure and Check state. During this state the UAS begins to configure itself; this means that all the services needed to accomplish the mission are deployed and configured over the hardware in the UAS. Finally, the services and its associated hardware are checked for correct functioning. Of course, all of these operations will be done with the UAS on ground still inside the hangar.

When all service are properly configured, the UAS can change to following state: parking. It is well known that the behavior of some subsystems can change when the engine is on, and in the parking state a new check is done after the engine is started. Some sensors, servo mechanisms and communication modules will be checked again when the engine is working. Also during this state we will monitor the engine parameters in order to start the mission in optimal engine conditions. If along this state we realized any anomaly in any service, the UAS comes back to the stop state. Rather, if the UAS pass all check procedures, the UAS can go on to taxi states.

Taxi States

At this moment, the UAS is working and needs to proceed to the runway to start the mission. The VAS can make this operation in two ways, as auto taxi and manual taxi. With the auto taxi the UAS look up an adequate runway and go to it by itself. This is a complex operation as it means that it automatically take into account air traffic controller interaction, wind estimation, etc. In other cases, the operator has to drive the UAS to the correct runway, in this situation the VAS is working in manual taxi mode. While the UAS is on ground the UAS speed will be limited. Also, if the UAS is in auto taxi and anything is wrong the VAS can give the control to the operator changing to manual taxi. When the UAS is in the runway heading then the VAS can pass to the take-off states. For airborne operation it is needed a flight plan loading into the VAS. If the VAS does not detect a loaded flight plan , it only permits manual manipulation.

Take-off States

After the taxi states the UAS is prepared to start the mission. The mission starts with the take off state; this operation is one of the most dangerous because the aircraft begins to fly and the UAS usually does not have enough altitude to response of any contingency.

This operation can also be develop manually or automatically. If the UAS takes off automatically and any contingency it happens the operator can take the UAS control, to solve the problem. However, if we have decided take-off manually we will remains in manual state, until the VAS can change to Waypoint Navigation state.

Navigation States

At this point of the diagram, the UAS is flying to a secure altitude. When the UAS has arrived to this altitude, it will change automatically to Waypoint Navigation which it is first state of the navigation group. The navigation states are composed by waypoint navigation state, directed state, hold at state and manual state.

In waypoint navigation the UAS follows the waypoints that have been uploaded previously in the VAS. While the UAS is in directed state it will maintain a specific altitude, airspeed and bearing. When the UAS is in Hold At state, it executes a hold pattern around the indicated waypoint. Finally, in manual state the operator has direct control over the airframe's control surfaces. The operator can switch from one Navigation state to each other as he commands from the ground station.

Landing States

When the mission has successfully finished, the UAS has to back home and prepare landing. In order to carry out this task the UAS switches to landing states. This group is composed of land pattern state, land abort state, auto land and manual land. During the land pattern state, the UAS will fly an approximation pattern in order to prepared for the landing. After this state we can choose between auto landing and manual landing.

In both cases, if some problem occurs during the landing, the UAS can switch to the land abort state. In this state the UAS climbs up to a secure altitude and goes back to the land pattern state to try to land again. If the landing has been achieved normally, the UAS will be braking on the runway. When the UAS speed is low enough, the UAS will switch to the Taxi state again and will continue to the hangar to finish the mission.

Safe States

If any failure occurs during the navigation states, the VAS can switch to the safe states. These states are composed by safe hold state, safe return state and safe land state. When we have a failure in the system, first of all the UAS change the state to the safe hold. In this state the UAS will remain trying to recovery the failure.

After a timeout the UAS will switch to Safe Return state. In this state the UAS will return to the closest emergency runway in order to start the pattern landing. After the pattern landing the UAS changes to Safe Land state. In this state a landing flight plan is generated. This flight plan is generated according to the

runway situation. These three states composed the safe states, however the system has another safe state: Safe Reaction.

Reaction State

The Safe Reaction is on charge of analyzing the environment around the aircraft and generating reactions in case of a quick evasive response is needed. When the Awareness services detects a dangerous situation, it generates an alarm and the VAS switch to Safe Reaction. In this state the UAS will be commanded by the Tactical Reaction service to solve the problem.

VI. Conclusions and Future Work

This paper presents an embedded architecture for Unmanned Aerial Vehicles which is on charge of the flight control but also of the execution of a target mission. The architecture is flexible and configurable because we have designed it to be usable in a large number of different civil missions as introduced in the paper. The cost and development time must be kept low to be market competitive. The proposed approach is based on Internet Service Oriented Architectures, which have shown to be easy to develop, maintain and integrate. The mission is resolved using distributed services, which may run in different hardware and which interact each other using a common middleware support. Efficiency, usability and quality of service are the main benefits of the middleware.

The main contributions of this work are the exhaustive listing of needed services and the abstraction layer constructed on top of them. We present a list of self-content services. Services are presented in a same group when they cooperate in the same main objective, such as flight or mission payload. But each service has a clear and specific task to execute inside the mission. Critical services may be redundant and the middleware will manage this redundancy in front of other services requirements. The second contribution is named UAV Service Abstraction Layer (USAL). The USAL is the distributed application interface that any UAV civil mission may need to use. Thus, the mission designer disposes of a distributed programming framework that includes a superset of data and functions needed for his implementation. Finally, a detailed example is presented for the autopilot, containing basically the public set of variables and events that this service offers, but also the operations that other services may require. A lateral benefit of the USAL is the unit normalization given.

Future work includes the final public interface description for all the services. In the same way that the Flight Control System Service has been detailed, we should detail the public variables, events, functions and files of every service presented. Also, for simple missions with few requirements of pay-load or flight conditions, the actual architecture components will be reduced to have competitive cost. The USAL should be able to overcome situations where some services are not present. Finally, a service allocation framework can be developed to help the mission designer in classifying, selecting and loading the several instances of services for a given mission in a semi-automated way.

References

[1] "EU Civil UAV Roadmap," http://www.uavnet.com.

[2] RTCA, "DO-304: Guidance Material and Considerations for Unmanned Aircraft Systems," March 2007.

[3] 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.

[4] D.C.Schmidt, "Middleware for Real-Time and Embedded Systems," *Communications of the ACM*, Jun 2002, pp. 43–48.

[5] Lopez, J., Royo, P., Pastor, E., Barrado, C., and Santamaria, E., "A Middleware Architecture for Unmanned Aircraft Avionics," *ACM/IFIP/USEUNIX 8th Int. Middleware Conference*, NewPort, California, Nov. 2007.

[6] W3C, "W3C Note: Web Services Architecture," http://www.w3c.org/TR/ws-arch.

[7] "UPnP Forum," http://www.upnp.org.

[8] Haiyang, C., Yongcan, C., and YangQuan, C., "Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey," *International Conference on Mechatronics and Automation (ICMA)*, IEEE, Harbin, China, 2007, pp. 3144–3149.

[9] Santamaria, E., Royo, P., Lopez, J., Barrado, C., Pastor, E., and Prats, X., "Increasing UAV capabilities through autopilot and flight plan abstraction," *Proceedings of the 26th Digital Avionics Systems Conference*, Dallas, Texas, 2007.

[10] "Guidance Material for the Design of Terminal Procedures for Area Navigation," European Organisation for the Safety of Air Navigation, 2003.