An Open Architecture for the Integration of UAV Civil Applications

E. Pastor, C. Barrado, P. Royo, J. Lopez and E. Santamaria
Dept. Computer Architecture, Technical University of Catalonia (UPC)
Spain

1. Introduction

The current Unmanned Aerial Vehicles (UAVs) technology offers feasible technical solutions for airframes, flight control, communications, and base stations. In addition, the evolution of technology is miniaturizing most sensors used in airborne applications. Hence, sensors like weather radars, SAR, multi spectrum line-scan devices, etc. in addition to visual and thermal cameras are being used as payload on board UAVs. As a result UAVs are slowly becoming efficient platforms that can be applied in scientific/commercial remote sensing applications (see Fig. 1 for the most common subsystems in an UAV).
UAVs may offer interesting benefits in terms of cost, flexibility, endurance, etc. Even remote sensing in dangerous situations due to extreme climatic conditions (wind, cold, heat) are now seen as possible because the human factor on board the airborne platform is no longer present. On the other side, the complexity of developing a full UAV-system tailored for remote sensing is limiting its practical application. Currently, only large organizations like NASA or NOAA have enough budget and infrastructure to develop such applications, and eventually may lease flight time to other organizations to conduct their experiments.
Even though the rapid evolution of UAV technology on airframes, autopilots, communications and payload, the generalized development of remote sensing applications is still limited by the absence of systems that support the development of the actual UAV sensing mission. Remote sensing 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, effective system support must be created to offer flexible and adaptable platforms for any application that is susceptible to use them.
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 UAV should be easy to define and flexible enough to adapt to the necessities of the mission. The payload of the UAV should be selected and controlled adequately. And finally, the mission should manage the different parts of the UAV application with little 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.

This research introduces a flexible and reusable hardware/software architecture designed to facilitate the development of UAV-based remote sensing applications. Over a set of embedded microprocessors (both in the UAV and the ground control station) we build a distributed embedded system connected by a local area network. Applications are developed following a service/subscription based software architecture. Each computation module may support multiple applications. Each application can create and subscribe to available services. Services can be discovered and consumed in a dynamic way like web services in the Internet domain. Applications can interchange information transparently from network topology, application implementation and actual data payload.

This flexibility is organized into a user-parameterizable *UAV Service Abstraction Layer* (*USAL*). The USAL defines a collection of standard services and their interrelations as a basic starting point for further development by users. Functionalities like enhanced flight-plans, a mission control engine, data storage, communications management, etc. are offered. Additional services can be included according to requirements, but all existing services and inter-service communication infrastructure can be exploited and tailored to specific needs. This approach reduces development times and risks, but at the same time gives the user higher levels of flexibility and permits the development of more ambitious applications.

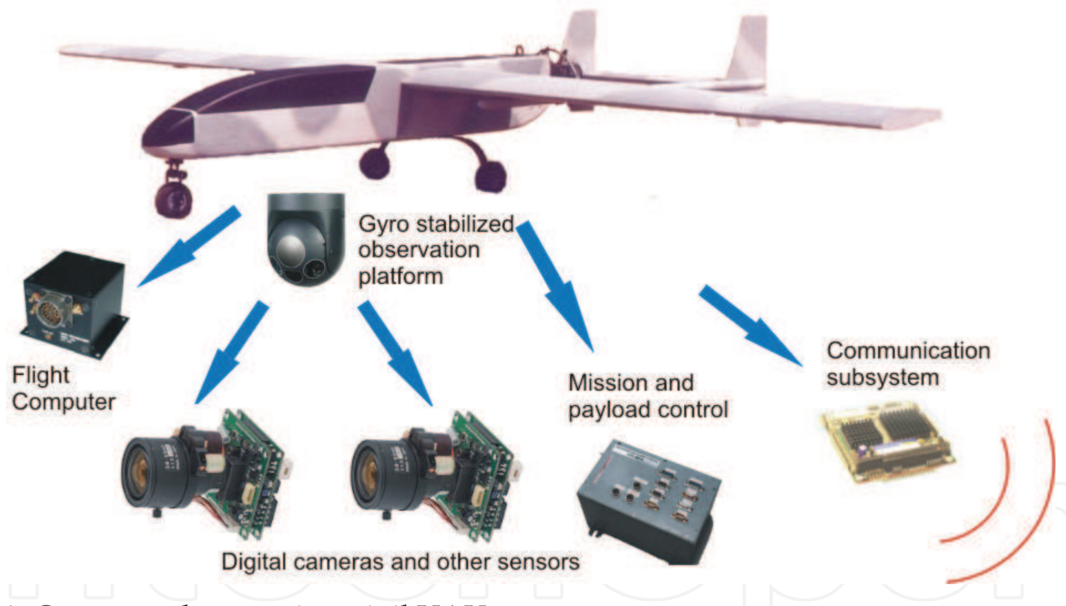

Gyro stabilized observation platform
Flight Computer
Mission and payload control
Communication subsystem
Digital cameras and other sensors

Figure 1. Common elements in a civil UAV system

This chapter is organized as follows. Section 2 generally describes the underlying service oriented technologies that will be applied to create the USAL. Section 3 overviews the architecture of the USAL and describes the most relevant services that are included in the USAL to facilitate the development of UAV applications. Section 4 details the *Virtual Autopilot System* (VAS) that permits the USAL architecture to abstract from autopilot details. Section 5 describes the Flight Plan Manager service that together with its RNAV based dynamic flight-plans constitute the core of the navigation capabilities inside the USAL. Finally, Section 6 concludes the chapter and outlines future research and development directions.

2. System Overview

This section describes the architecture we propose for executing UAV 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, which will implement the required functionalities. These services cooperate for the accomplishment of the UAV mission. They rely on a middleware (Lopez et al. 2007) 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.

2.1 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 is 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 UAV, the addition of new nodes can be hot plugged to the LAN with little effort. The system can use wake-on-LAN capabilities to switch on/off a node when required, at specific points of the mission development.

2.2 Service Oriented Approach

Service Oriented Architecture (SOA) is getting common in several domains. For example, Web Services in the Internet world (W3C, 2004), and Universal Plug and Play (UPnP, 2008) in the home automation area. The main goal of SOA is to achieve loose coupling among interacting components. We name the distributed components 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 are 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 by 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 a service needs some external functionality, it asks the network 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.

2.3 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 handles their discovery. The middleware also handles all the transfer chores: message addressing, data marshalling 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 behaviour.
- *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.

Fig. 2 shows the proposed UAV distributed architecture. 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 Module.

Each node of the UAV distributed architecture executes also a copy of the Service Container software (see Fig. 2). 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 point of view the middleware builds a global LAN network that connects the LAN on ground and the LAN on board.

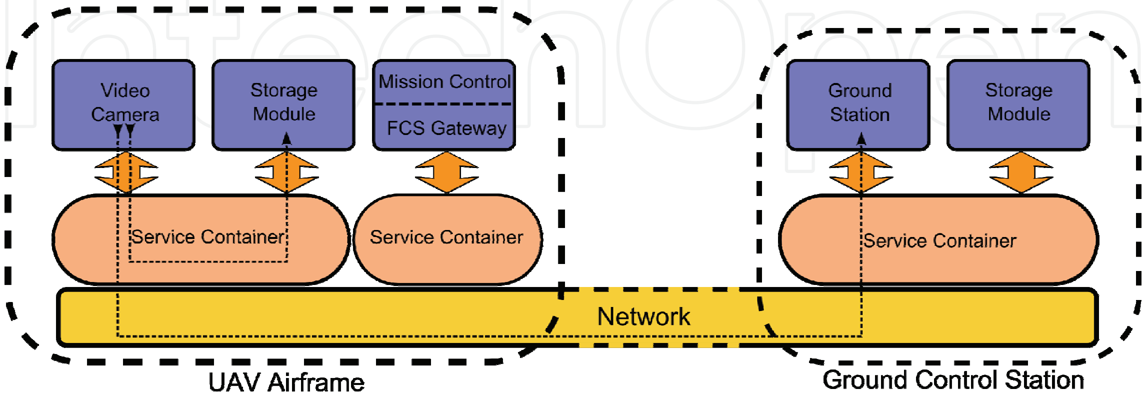


Figure 2. Overview of the architecture implementing the underlying middleware

2.4 Communication Primitives

For the specific characteristics of a UAV 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 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, 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.

3. USAL: UAV Service 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 UAV may alleviate the development 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 UAV (mini or tactical UAV in our case) and the mission objectives. From the study and definition of several UAV missions, one can identify the most common requirements and functionalities that are present among them (UAVNET, 2005; RTCA, 2007; SC-203, 2007; Cox et. al., 2006).
These common requirements have been identified and organized leading to the definition of an abstraction layer called UAV Service Abstraction Layer (USAL). The goal of the USAL is twofold (Pastor et. al., 2007; Royo et. al., 2008):

- Reduce "time to marked" when creating a new UAV 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.
- Simplify the development of all systems required to develop the actual mission assigned to the UAV. 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 UAV 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.

3.1 USAL Service Types

Even though the USAL is composed of a large set of available services, not all of them have to be present in every UAV or in any mission. Only those services required for a given configuration/mission should be present and/or activated in the UAV. Available services have been classified in four categories according to the requirements that have been identified (see Fig. 3).
The principal element is the computing system that should orchestrate the overall mission. This system may be extended with additional systems specific to the mission like image processing hardware accelerators, etc. Storage and communication management should be also included by default. This set of standard plus user-defined services that control the mission intelligence are named *Mission Services*.

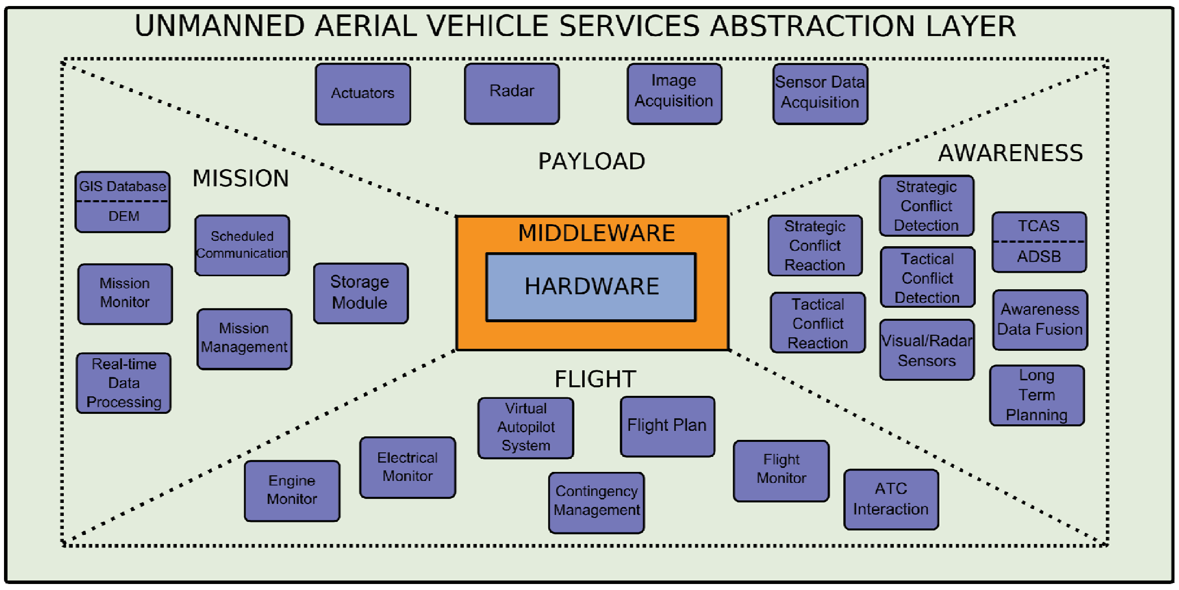


Figure 3. Overview of the USAL service-based architecture

The next relevant system is the UAV autopilot. USAL considers the autopilot as a co-processor; it provides the system with a specific set of primitives that control the flight in the short term. The autopilot operation is supervised by a Flight Plan Manager that abstracts users from autopilot peculiarities and offers flight plan specifications beyond classical way point navigation, thus improving operational capabilities. Additional services help improving the safety and reliability of the operation. The services in charge of the flying capabilities of the UAV are named *Flight Services*.

Successful integration of UAV in non-segregated aerospace will require a number of features to be included in the UAV architecture. Interaction with cooperative aircrafts through transponders, TCAS or ADS systems; and detection of non-cooperative aircrafts through visual sensors, should be implemented and the UAV must inform the pilot in command or automatically react following the operational flight rules for UAV that are currently being developed (EUROCONTROL, 2003; FAA, 2008). However, for certain cases, e.g. flying in segregated airspace, such services may not be necessary. Services that manage the interaction of the UAV with the surrounding airspace users, air traffic management or conditions are named *Awareness Services*.

Payload includes all those other systems carried on board the UAV. We divide them in data acquisition systems (or input devices) and actuators (or output devices). Input devices can be flight sensors (GPS, IMU, Anemometers) and earth/atmosphere observation sensors (visual, infra-red and radiometric cameras, chemical and temperature sensors, radars, etc.) Output devices are few or even do not exist in UAV civil missions because of the weight limitations: flares, parachutes or loom shuttles are examples of UAV actuators. Services controlling these devices are named *Payload Services*.

3.2 Mission Services

From the end-users point of view, earth observation is the main target of a UAV flight. For this reason the user selects a geographic area that constitutes the initial objective of the observation. Also, the user must define the input sensors to activate and the conditions for mission updates.

The DO-304 RTCA (RTCA, 2007) describes 12 scenarios, selected from a list of 70 as representative of UAV missions. Scenarios are representative of different types of airframes and navigation conditions. Regarding their navigation procedures, we have classified them in three types, with increasing complexity:

- Static Area Surveillance. The navigation pattern over the surveillance area is given before take-off. This navigation is the base of the following RTCA scenarios: Communicator Repeater (Scenario 7), Courier (Scenario 8), Border Surveillance (Scenario 16), Coastal Border Control (Scenario 26), High Altitude Communication Relay (Scenario 40).
- Target Discovering. The UAV System has a recognition service that can detect on-the-fly some objects or behaviours in the static area of surveillance. This navigation is the base of the following RTCA scenarios: Perimeter Defence (Scenario 3), OAV Police Operation (Scenario 10), and Mass Casualty Analysis (Scenario 29). Although in some scenarios the mission continues after target discovering, the mission intelligence is relegated to ground operators decision and control.
- Dynamic Target Tracking. The most advanced missions are those that assume intelligence Mission Services, with the capacity to redefine the surveillance area with no human intervention. Hurricane Chase (Scenario 2), Coast Guard Reconnaissance and Surveillance (Scenario 37) are scenarios where this situation is given at some level.

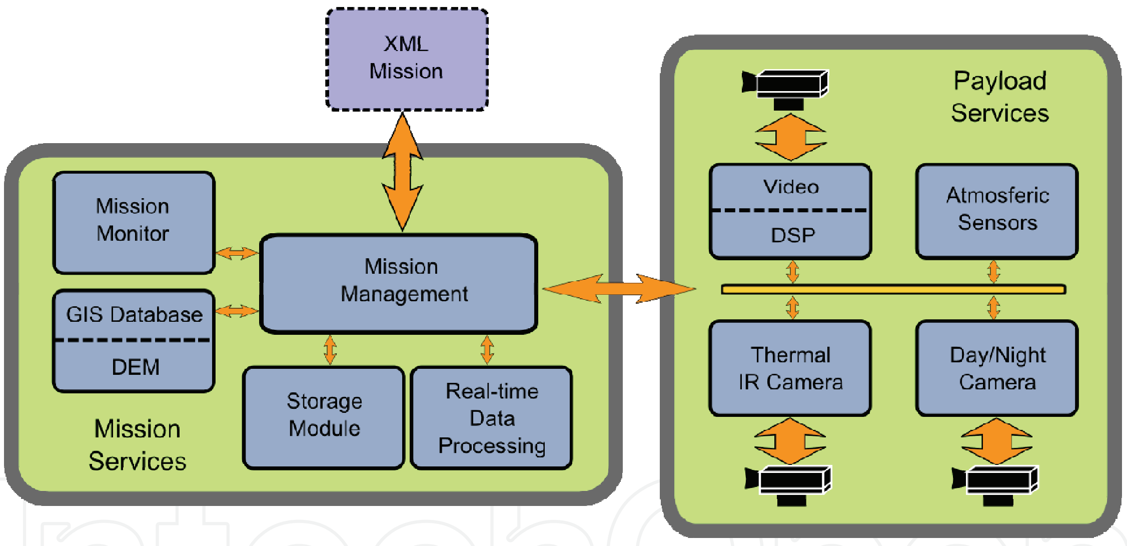


Figure 4. Overview of the available mission and payload service category

In general, the most complex missions include the simplest missions as part of its objective. For example, in order to track a moving object, its previous discovering is needed, which is based on an area surveillance mission. For the Static Area Surveillance the only condition needed is the end-of-mission condition, but for the other two the end-users have to give the condition for Target Recognition.

The USAL offers a number of predefined services to implement a wide range of missions, namely the *Mission Manager*, the *Real-Time Data Processing*, the *Storage*, the *Scheduled Communications*, the *GIS/DEM Database* and *Mission Monitor* (see Fig. 4). These services can be adapted to requirements by means of parameters (e.g. specific flight plan, sensors to be activated, information flows, etc), by adding specific software code to be executed or by adding specific user defined services. Next we describe in more detail some of them.

The *Mission Manager* (MMA) 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 MMA 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 MMA is capable of modifying the actual flight plan by redefining its parameters or by defining new stages or legs.
The *Real-Time Data Processing* (RDP) gives the intelligence for complex missions. The RDP offers predefined image processing operations (accelerated by FPGA hardware if available) that should allow the MMA to take dynamic decisions according to the actual acquired information.
The *Mission Monitor* (MMO) shows to end-users human friendly useful information about the mission. For example, during a wildland fire monitoring mission, it may present the current state of the fire front over a map. The MMO is executed on the ground and should be highly parameterized to fit the specific requirements of each mission.

3.3 Flight Services

Many autopilot manufacturers are available in the commercial market for tactical UAVs with a wide variety of selected sensors, sizes, control algorithms and operational capabilities. However, selecting the right autopilot to be integrated in a given UAV 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 UAV.
Current commercial UAV autopilots also have two clearly identified drawbacks that limit their effective integration with the mission and payload control inside the UAV:

- Exploiting the on-board autopilot telemetry by other applications is complex and autopilot dependent. Autopilot’s telemetry is typically designed just to keep the UAV state and position under control and not to be used by third party applications.
- The flight plan definition available in most autopilots is just a collection of waypoints statically defined or hand-manipulated by the UAV’s operator. However, no possible interaction exists between the flight-plan and the actual mission and payload operated by the UAV.

Flight services are a set of USAL applications designed to properly link the selected UAV autopilot with the rest of the UAV avionics (Santamaria et al., 2008; Santamaria et al., 2007), namely the *Virtual Autopilot Service*, the *Flight Plan Manager Service*, the *Contingency Service*, the *Flight Monitor Service*, etc. (see Fig. 5).
The *Virtual Autopilot Service* (VAS) is a system that on one side interacts with the selected autopilot and is adapted to its peculiarities. VAS abstracts the implementation details from actual autopilot users. From the mission/payload subsystems point of view, VAS is a service provider that offers a number of standardized information flows independent of the actual autopilot being used.
The *Flight Plan Manager* (FPM) is a service designed to implement much richer flight-plan capabilities on top of the available autopilot capabilities. The FPM 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. These legs can be modified by other services in the USAL by changing the configuration parameters without having to redesign the actual flight-plan; thus enabling the easy cooperation between the autopilot and the UAV mission.

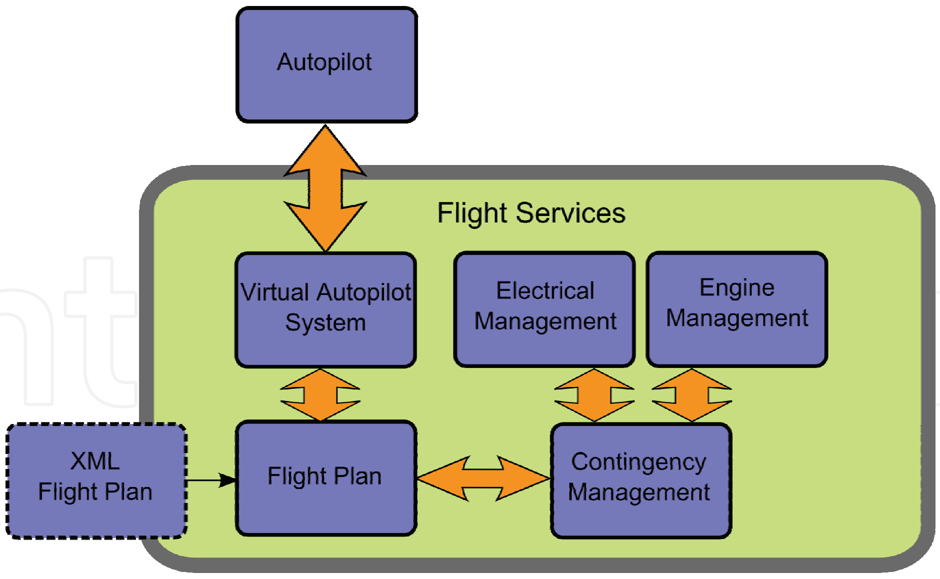


Figure 5. Overview of the available flight service category

The *Contingency Management* services are a set of services designed to monitor critical parameters of the operation (like battery live, fuel, flight time, system status, etc.). In case contingencies are detected, actions will be taken in order to preserve the safety and integrity of the UAV: from flight termination, mission abort or system re-cycle.

3.4 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 flying at any altitude, where VFR aircrafts are found.
UAVs must rely 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 for such functionalities. Flight Services are in charge of the aircraft management in normal conditions while the Awareness Services are in charge of monitoring surrounding situations and overtake aircraft management in critical conditions. In this case mission services come to a second priority, until flight conditions become again normal. The list of awareness services is seen in Fig. 6.
The *Awareness Data Fusion* (ADF) is a service designed to collect all available data about air vehicles surrounding our UAV, terrain and meteorological conditions. All this information can be obtained either by on board sensors or even through an external provider.
The *Tactical/Strategic Conflict Detection* services will analyze the fused information offered by the ADF in order to detect potential collision conflicts with objects/terrain/bad climate. Depending on the type of conflict, different types of reaction procedures will be activated. While reaction is executed the ADF will keep monitoring that the conflict is really being avoided.
The *Tactical/Strategic Reaction* services, will implement avoidance procedures according to the severity of the conflict. Tactical reaction is designed in such a way it can overtake the FPM in order to execute a radical avoidance manoeuvre. Once completed, the FPM will retake the control. A strategic reaction will command the FPM to slightly modify its selected

flight plan trying to avoid the conflict but at the same time retaining the original mission requested by the Mission Manager.

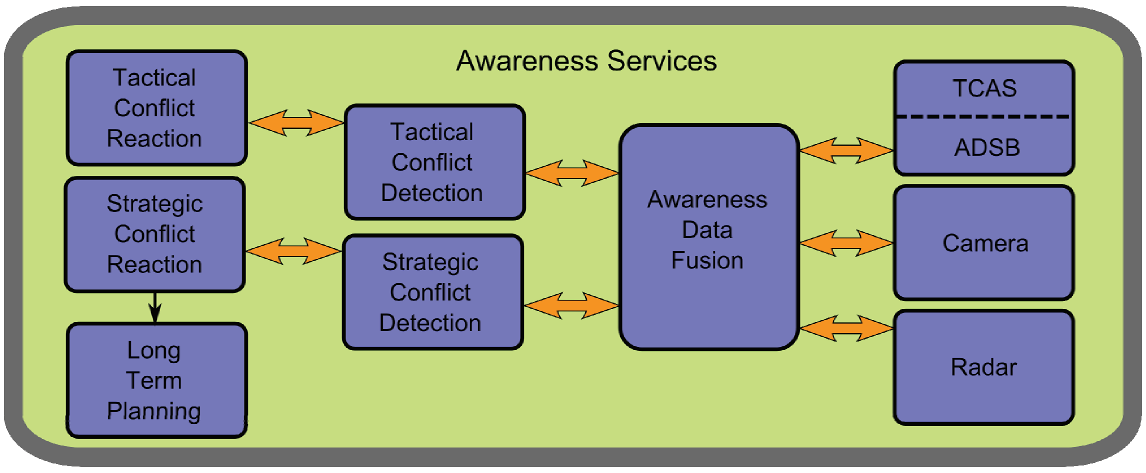

Awareness Services
Tactical Conflict Reaction
Strategic Conflict Reaction
Long Term Planning
Tactical Conflict Detection
Strategic Conflict Detection
Awareness Data Fusion
TCAS
ADSB
Camera
Radar

Figure 6. Overview of the available flight service category

3.5 Payload Services

Payload services are defined for sensor 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.

4. The Virtual Autopilot System (VAS)

The Virtual Autopilot Service is the component that will interact between the autopilot and the rest of the components of the USAL (Royo et. al., 2008). After studying several UAV autopilots we have seen that their functioning and capabilities are very similar, however their implementation details greatly differ (Haiyang et al., 2007). To improve the flexibility of a UAV it 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 UAV mission needs to control autonomously the aircraft following a list of waypoints, the VAS clearly belongs to the minimum services required in the USAL.

4.1 VAS Functional Overview

The VAS is specially suited to work in conjunction with the Flight Plan Manager service (FPM). The latter service stores the flight plan of the mission which is composed of a complex hierarchy of waypoints that the UAV must fly. The VAS will command the hardware autopilot to guide the UAV flight and it will ask the FPM 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 UAV.
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 current missions or applications being developed by the UAV using a simple waypoint-based flight plan may be too restrictive. In addition to the VAS the Flight Plan Manager service (FPM) is designed to implement much richer flight-plan capabilities than offered by the actual autopilot. The FPM 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. Fig. 5 provides a schematic view of the relation between VAS and FPM inside the USAL.

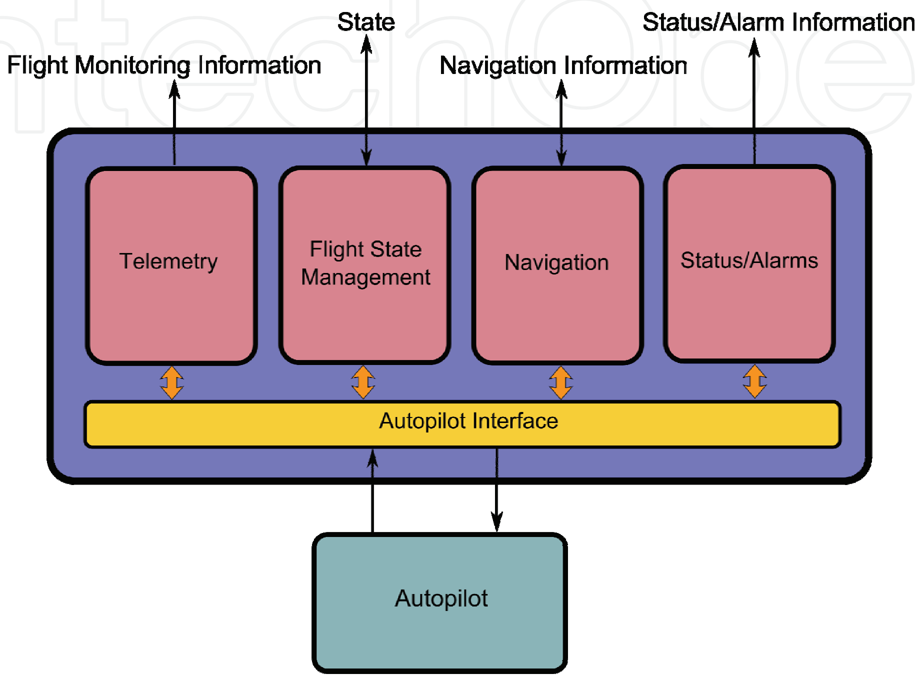


Figure 7. Information flow inside the Virtual Autopilot System

Given that the real autopilot capabilities will be much simpler than those available in the FPM, 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 FPM 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 FPM, we obtain a highly capable platform that can be easily integrated to perform much more efficiently a number of valuable missions.

4.2 VAS Architecture

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 UAV, the VAS is a service provider that offers a number of information flows to be exploited. 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 autopilot being used. VAS belongs to the set of services defined in the 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 UAV 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.
UAV 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. 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. Fig. 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. These four groups of interfaces are described more in detail in the following subsections.

4.3 Flight Monitoring Services

There are several ways to expose the information generated by the sensors of the autopilot. Usually, the autopilot manufacturer groups all this information in large packets of data, which 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. The remainder of the section shows all the Flight Management Information which the VAS will publish the rest of the services on the network.

- 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 wind direction estimation around the system. North, east and downwind vector is offered in meters per second.
- Mission Time: Indicates the current mission time in seconds since the VAS start-up or the last mission time reset.

In general, UAV autopilots provide lots of information from all its 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.

4.4 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 internal waypoints as it consumes system waypoints and commands.
The next group of information is the Output Service 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:

- Current Waypoint: Indicates the current waypoint where the system is flying to. 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 preferred runway (not enough fuel, battery, structural damage, etc.).
- UAV Direction: Indicates selected direction of the UAV 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 are included latter in the section. State is reported each time the VAS switches from one state to another; and each time a state change is requested but cannot be fulfilled.

All of this service information will be mainly used for the FPM in order to interact dynamically with VAS and the mission services. In case the FPM or some other service related to the UAV flight fails, the VAS can take control of the UAV. This emergency state implies a change in the flight state management to Safe mode until the flight plan is recovered or the UAV 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.
The next group of information is the Input Navigation Service 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:

- 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.
- 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 which the UAV will fly.
- UAV Speed: Set target indicated airspeed. This packet only has effect once in directed state. However, each waypoint defines the speed which the UAV will have to arrive over that waypoint.
- UAV Altitude: Set target altitude. As UAV speed, this packet only have effect once in directed state and each waypoint defines the altitude which the UAV will have to fly over that waypoint.
- New Main Runway: Set coordinates of runway. The main runway is where the UAV will land. In principle the main runway is where the UAV will take off. However, the UAV will sometimes need a closer runway for landing, especially if it has happened any contingency and the UAV needs to land in order to solve the problem.
- New Alternative Runway: Set coordinates of alternative runway. In the alternative runway, the UAV will save alternative places where the UAV can land. The purpose of this packet is to store a runway closer to where the UAV is flying each moment.
- Change VAS Mode: Sets the current VAS state. With this packet some services as FPM, ground station service or awareness service can switch the VAS states.
- 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 a 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 FPM controls the entire mission flight plan. In some cases, the FPM may be interested in discard a range of waypoints of the flight plan. These waypoints can take several legs or can be part of a leg.

4.5 VAS and Autopilot Status/Alarms

The VAS is the interface between the autopilot and the rest of the system. So it is in charge of informing the status of the autopilot and its own status. An 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 devices has a failure the VAS will send an alarm to the network. All of these alarms are sent 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 periodical information like the telemetry flows.
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). Another important situation that should be controlled is the voltage of the different hardware components of the UAV. Low voltages clearly indicate 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 problems in the different sensors inside the autopilot (Accelerometer, Gyroscope, Magnetometer, Anemometer and Pressure altimeter). UAV operation can usually continue however on a degraded mode. In this case, the UAV 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 FPM (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.
Another specific situation that is notified by the UAV, 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 UAV 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 abnormal behaviour in its internal calculations.

4.6 VAS Operational States

During its operation an autopilot has different forms of guidance: take-off, waypoint navigation, directed flight, landing, etc. Depending on the implementation and the capabilities of the selected autopilot, these behaviours 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. Commercial autopilots are much focused on flight states, however a mission can be composed of many different states, for example, and we can have different behaviours 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 UAV 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 fulfil 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.
Fig. 8 shows all the VAS states. This is a graph diagram in which each node corresponds to one state. In each state, the UAV develops several tasks in order to achieve the goals of the mission. 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.

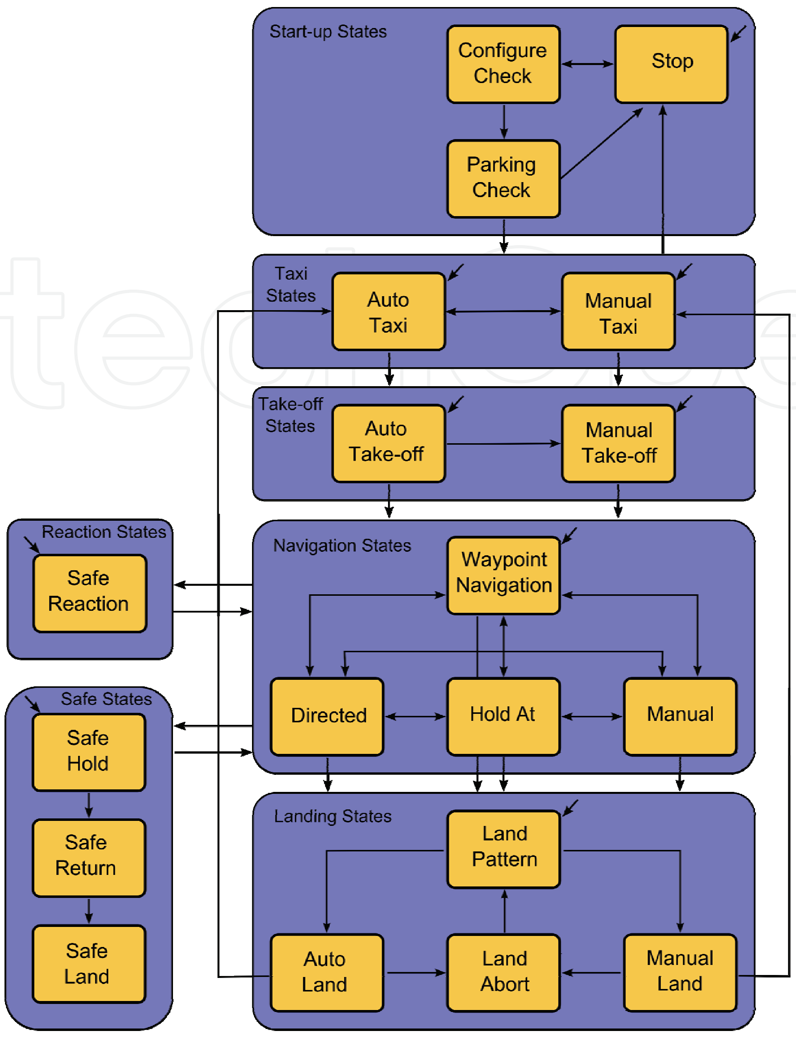


Figure 8. VAS Operational States and Transitions

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

When all services are properly configured, the UAV can change to the following state: Parking & Check. It is well known that the behaviour 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 detect any anomaly in any service, the UAV comes back to the Stop state. If the UAV pass all check procedures, the UAV can move to taxi states.

Taxi States. At this moment, the UAV is ready and may proceed to the runway to start the flight. The VAS can execute this operation in two ways, as auto taxi or as manual taxi. With the auto taxi the UAV look up an adequate runway and goes to it by itself. This is a complex

operation as it means that it automatically takes into account air traffic controller interaction, wind estimation, etc. In other cases, the operator has to drive the UAV to the correct runway, in this situation the VAS is working in manual taxi mode.
While the UAV is on ground the UAV speed will be limited. Also, if the UAV is in auto taxi and anything is wrong the VAS can give the control to the operator changing to manual taxi. When the UAV is in the runway heading the VAS can pass to the take-off states. For airborne operation a flight plan it is needed 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 UAV 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 UAV usually does not have enough altitude to respond to any contingency.
This operation can also be developed manually or automatically. If the UAV takes off automatically and any contingency happens the operator can take the UAV control, to solve the problem. However, if we have decided to take-off manually we will remain in manual state, until the VAS can change to Waypoint Navigation state.
Navigation States. At this point of the diagram, the UAV is flying to a secure altitude. When the UAV has arrived to this altitude, it will change automatically to Waypoint Navigation which is the 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 UAV follows the waypoints that have been uploaded previously in the VAS. While the UAV is in directed state it will maintain a specific altitude, airspeed and bearing. When the UAV 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 been successfully finished, the UAV has to go back home and prepare for landing. In order to carry out this task the UAV switches to landing states. This group is composed of Land Pattern state, Land Abort state, Auto Land and Manual Land states. During the the Land Pattern state, the UAV will fly an approximation pattern in order to prepare 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 UAV can switch to the Land Abort state. In this state the UAV 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 UAV will be braking on the runway. When the UAV speed is low enough, the UAV 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, the UAV will change the state to the Safe Hold. In this state the UAV will remain trying to recover from the failure.
After a timeout the UAV will switch to Safe Return state. In this state the UAV will return to the closest emergency runway in order to start the landing pattern. After the landing pattern the UAV will change to the 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 compose the safe states; however the system has another safe state: Safe Reaction.

Safe 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 detect a dangerous situation, they generate an alarm and the VAS switches to Safe Reaction. In this state the UAV will be commanded by the Tactical Reaction service to solve the problem.

5. Mission Aware Flight Planning

Previous sections have introduced the proposed UAV architecture. In this section we detail the specification mechanism that will be used to describe the UAV flight plans. Most current UAV autopilot systems rely on lists of waypoints as the mechanism for flight plan specification and execution. This approach has several important limitations: (1) It is difficult to specify complex trajectories and it does not support constructs such as forks or iterations. (2) It is not flexible because small changes may imply having to deal with a considerable amount of waypoints and (3) it is unable to adapt to mission circumstances. Besides (4) it lacks constructs for grouping and reusing flight plan fragments. In short, current autopilots specialize in low level flight control and navigation is limited to very basic go to waypoint commands.
We believe that it is necessary to improve current UAV operation with higher level constructs, with richer semantics, and which enable flight progress to be aware of mission variables must be introduced. For that reason a new flight plan specification mechanism is introduced.

5.1 Flight Plan Overview

This section describes the major characteristics of the specification mechanism that will be used to program the FPM service. Some ideas are based on current practices in commercial aviation industry for the specification of RNAV procedures (EUROCONTROL, 2003; FAA, 2008) which is briefly described in the next paragraphs.
Radio Area Navigation (RNAV) is a method of navigation that takes advantage of the increasing amount of navigation aids (including satellite navigation) and permits aircraft operation on any desired flight path. RNAV procedures are composed of a series of smaller parts called legs. To translate RNAV procedures into a code suitable for navigation systems the industry has developed the “Path and Termination” concept. Path Terminator codes should be used to define each leg of an RNAV procedure. Leg types are identified by a two letter code that describes the path (e.g., heading, course, track, etc.) and the termination point (e.g., the path terminates at an altitude, distance, fix, etc.).
Our specification mechanism makes use of the Path Terminator concept to describe basic legs. A subset of RNAV legs applicable to GPS navigation is also of interest. These elements are brought to the UAV field and extended with additional constructs. New control constructs such as iterative legs and intersection legs are added. Reverse traversal of legs belonging to an iterative construct is supported. And adaptivity is increased by means of parametric legs. Further details are given in the next subsections, which describe the flight plan structure and its elements.
The flight plan represents the instructions that will be given to the FPM. A flight plan follows a hierarchical structure and is composed of stages, legs and waypoints (see Fig. 9).
Stages are the largest building blocks within a flight plan. They organize legs into different phases that will be performed in sequence. Legs specify the path that the plane must follow in order to reach a destination waypoint from the previous one. Several primitives for leg specification are available.

A waypoint is a geographical position defined in terms of latitude/longitude coordinates. Waypoints can be named or unnamed depending on whether they are associated to a fix. A fix corresponds to a geographical position of interest with a name and description. Another distinction is made between fly-by and fly-over waypoints. Fly-by waypoints will be used when the aircraft should start turning before reaching the waypoint. Fly-over waypoints require the aircraft to fly over them before initiating a turn. A waypoint may also be accompanied by altitude and speed change indications.
Optionally, a partial flight plan to be carried out if an emergency occurs can be associated to a flight plan. This emergency plan will be superseded by emergency plans specified at stage or leg level. In this way the user will be able to choose the appropriate level of granularity for alternate plans specification. A partial flight plan follows the same structure as a normal flight plan but contains only those stages necessary to fly from the current position to the landing runway of choice.

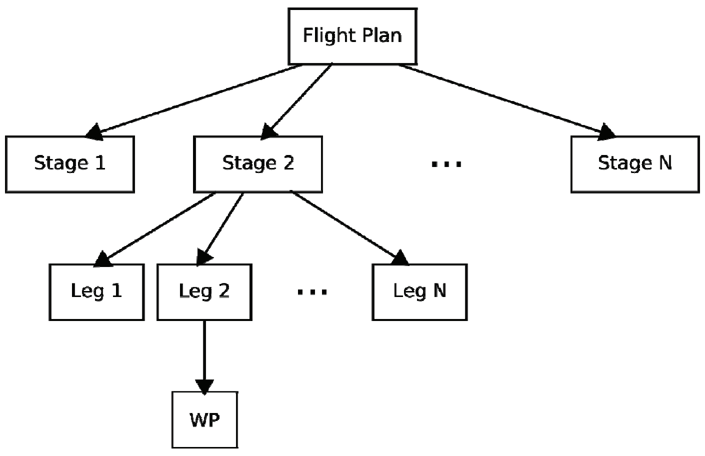


Figure 9. Overview of the Flight Plan Hierarchy

Stages constitute high-level building blocks for flight plan specification and are used to group together legs that seek a common purpose. They correspond to flight phases that will be sequentially executed. Fig. 10 shows a complete flight plan with all stages in the context of a fire fighting mission. The flight plan starts with a Take-Off. Once the aircraft is airborne it will perform a Departure Procedure that will connect with an En Route leading to the forest fire site. Upon arrival the Mission stage will start. When this stage concludes the UAV will enter another En Route stage leading to the landing area. There an Arrival Procedure will be executed to connect with the final Approach and Land stages. Each one of these stages will be composed of a set of legs as described in the following paragraphs.

5.2 Detailed Leg description

A *leg* specifies the flight path to get to a given waypoint. In general, legs contain a destination waypoint and a reference to their next. Most times legs will be flown in a single direction, but within iterative legs reverse traversal is also supported. There are four different kinds of legs:

- *Basic legs*: Specify leg primitives such as "Direct to a Fix", "Track to a Fix", etc.
- *Iterative legs*: Allow for specifying repetitive sequences.
- *Intersection legs*: Provide a junction point for legs which end at the same waypoint, or a forking point where a decision on what leg to fly next can be made.

- *Parametric legs*: Specify legs whose trajectory can be computed given the parameters of a generating algorithm, e.g. a scan pattern.

Basic Legs. A number of basic legs are available to the flight plan designer. They are referred to as basic legs to differentiate them from control structures like iterative or intersection legs and parametric legs. All of them are based on already existing ones in RNAV. Its original name is preserved.

- *Initial Fix* (IFLeg): Determines an initial point. It is used in conjunction with another leg type (e.g. TF) to define a desired track.
- *Track to a Fix* (TFLeg): Corresponds to a straight trajectory from waypoint to waypoint. Initial waypoint is the destination waypoint of the previous leg.
- *Direct to a Fix* (DFLeg): Is a path described by an aircraft's track from an initial area direct to the next waypoint, i.e. fly directly to the destination waypoint whatever the current position is.
- *Radius to a Fix* (RFLeg): Is defined as a constant radius circular path around a defined turn center that terminates at a waypoint. It is characterized by its turn center and turn direction.
- *Holding Pattern*: Specifies a holding pattern path. There are three kinds of holding patterns: Hold to an Altitude (HALeg), Hold to a Fix (HFLeg) and Hold to a Condition (HCLeg). In all cases the initial waypoint, the course (azimuth) of the holding pattern and the turn direction must be specified. The distance between both turn centers and the diameter of the turn segments is also needed.

The three available holding types differ in how they are terminated. Hold to an Altitude terminates when a given altitude is reached, therefore the target altitude and the climb rate must be indicated. A Hold to a Fix is used to define a holding pattern path, which terminates at the first crossing of the hold waypoint after the holding entry procedure has been performed. The final possible type is the Hold to a Condition. In this case the holding pattern will be terminated after a given number of iterations or when a given condition no longer holds (regardless of the number of iterations).

Iterative Legs. A complex trajectory may involve iteration, thus the inclusion of iterative legs. An iterative leg has a single entry (i.e. its body can be entered from a single leg), a single exit and includes a list with the legs that form its body. Every time the final leg is executed an iteration counter will be incremented. When a given count is reached or a specified condition no longer holds the leg will be abandoned proceeding to the next one.

Intersection Legs. Intersection legs indicate points where two or more different paths meet and where decisions on what to do next can be made. All joins and forks will end and start at an intersection leg.

Parametric Legs. In many occasions the dynamic characteristics of the mission environment will make previous leg types insufficient. Parametric legs provide an increased level of adaption to changes that occur during mission time. With parametric legs the flight path is dynamically generated according to input values. Eventually a library of different parametric legs will be available, complete enough so that a wide range of missions can be performed. With the use of parametric legs two goals are achieved. First, complex trajectories can be generated with no need to specify a possibly quite long list of legs. Second, the UAV path can dynamically adapt according to the input values.

Conditions. There are several points in the flight plan where conditions can be found, namely in holding patterns, iterative and intersection legs. For intersection legs, they are necessary in order to determine what path to follow next. For the rest of legs they will let the

FPM know when to leave the current leg and proceed to the next one. Conditions will not be directly specified in the flight plan. Instead, each leg that depends on a condition will contain a reference which will be used to identify the condition. Conditions will be stored and processed separately. When the outcome of a condition is set the FPM will be notified so that this change is taken into account for waypoint generation.
Conditions can be based on elapsed flight time, whether some task has been completed, on counters and on operator input. Operator input will always override any automatically generated outcome. In case of conflict or if the condition cannot be resolved we will resort to the default value or await user input.

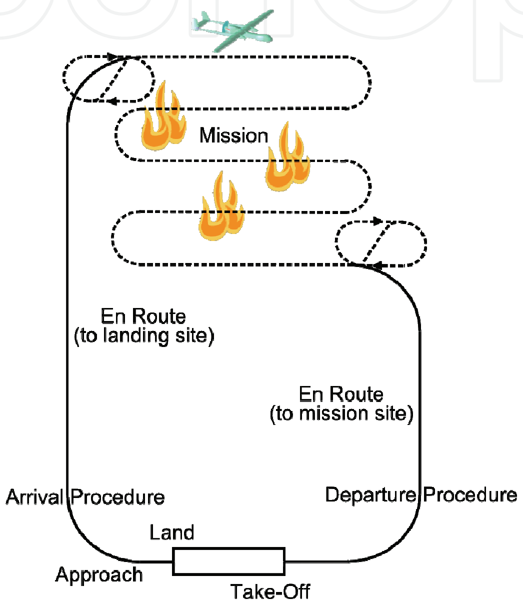


Figure 10. Flight Plan Application Example

5.3 Flight Plan Manager

This section presents the USAL service responsible for processing and executing the proposed flight plans: the Flight Plan Manager service (FPM). The FPM forms part of a wider collection of services that provide the UAV with all its capabilities. The FPM belongs to the flight services category, it will collaborate with other on-board and remote services in order to execute the given flight plan.
The FPM is responsible for executing flight plans, but it doesn't operate in a stand-alone fashion. To achieve its goals it collaborates with other services. The main service the FPM collaborates with is the Virtual Autopilot Service (VAS), which is the only service with direct access to the installed autopilot. Apart from the VAS, the most relevant services the FPM interacts with are the Ground Control Station (GCS) and the Mission Management service (MMA). The GCS provides monitoring and control capabilities to a human operator. The Mission Management Service is in charge of evaluating conditions and updating parametric legs.
In order to execute the flight plan the FPM will send navigation commands to the VAS. These commands mainly consist in waypoints the aircraft has to fly to. Since the flight plan is specified in terms of legs some translation process is needed for converting them into the waypoint sequences expected by the VAS.

Computing waypoint sequences that approximate the legs specified in the flight plan is the main task of the FPM execution engine. This flow of waypoint commands is the main form of interaction between the FPM and the VAS. Sometimes, these waypoints will be accompanied by additional fields indicating speed and altitude change requests.
Other commands related to waypoint management include clearing all sent but pending waypoints. This will be necessary, for instance, when an emergency which forces to execute an alternative plan occurs. The FPM will also issue partial cancellation commands as would be the case when ignoring a number of waypoints in order to directly jump to a given leg. Another type of command will allow the FPMS to change the VAS operation mode to request special operations such as taking-off or landing.
The Ground Control Station is the interface to the system that human operators will interact with. As such it must be able to manage and control several aspects of the flight plan and its execution. To this end, the FPM provides the following operations:

- Load flight plan: Load the flight plan. Before starting any mission a flight plan must be submitted to the UAV.
- Set initial leg: Since each stage of the flight plan can store multiple paths, when there is more than one possibility for the first flight plan stage, the GCS operator will indicate which one to start with.
- Start: Initiate aircraft flight. An automatic or manual take-off, depending on UAV capabilities and configuration will be performed.
- Pause: The aircraft will fly a holding pattern until commanded to resume flight plan execution.
- Manual: Inform the FPM that we are going into manual mode.
- Resume: Switch from paused or manual mode to normal automatic operation.
- Stop: Stops FPM operation.
- Goto leg: Fly directly to the given leg skipping intermediate ones without abandoning automatic mode of operation.
- Update flight plan: Update command provided for modifying the flight plan. A waypoint can be moved, the values of a parametric leg can be updated and other leg parameters such as speed and altitude can be changed.
- Set condition result: Set the result of any of the conditions the flight plan depends on.
- Trigger emergency return: The FPM will switch to the emergency flight plan defined for the current leg, stage or flight plan.

Apart from the listed operations, the FPM also provides a number of information flows that enable monitoring. This data consists in the position of the aircraft in flight plan terms, i.e. what are the current stage, leg and other leg-related information such as current iteration of an iterative leg, etc. It also provides information about the current operating state of the service.
One of the main features of our flight plan specification mechanism is that it enables the UAV to adapt to mission circumstances. This can be done in two manners: first, with the possibility of using conditions in different leg types. Second, by using parametric legs, whose final form depends on the values of the input parameters.
Conditions mark a point where a decision can be made about what path to follow next. The decision-making process is not directly done by the FPM. The flight plan contains references to condition identifiers. The outcome of these conditions will be set by the Mission Management Service. The MMA will also decide what the actual parameter values for parametric legs are. These functions are currently done from the GCS but the inclusion of the MMA will provide the UAV with a high degree of automation.

The main task of the FPM consists in generating the sequence of waypoints that will direct the UAV flight. At the same time the FPM has to respond to commands sent from the GCS and be aware of situations where VAS control is taken over by other services. This results in the FPM operating in a number of states as seen in Fig. 11. A description of each one of these states follows:

- *On Command*: The FPM can be either on command or on standby. If on command it has control over the VAS and determines the path followed by the aircraft. When a contingency occurs the main flight plan is replaced by the corresponding emergency plan and waypoint generation starts over. If the flight plan is updated all non-flown waypoints affected by the change will be cancelled and replaced by new ones. *On Command* is a super-state that encompasses two sub states, namely *Auto* and *Paused*.
- *Auto*: When in this state the FPM is generating and forwarding waypoints to the VAS.
- *Paused*: The FPM has received a pause command. This directly translates to issuing a change of state command to the VAS requiring it to perform a holding pattern. When the resume command is received it will notify the VAS that waypoint navigation can continue.
- *On Standby*: This state is entered when the FPM is notified that some other service has taken control over the VAS. It can be entered because another service is trying to avoid a collision or because the VAS is now under manual control. It differs from the *Paused* state in that the FPM is not going to send any message to the VAS. It just waits and tries to recover and continue flight execution once it regains control.

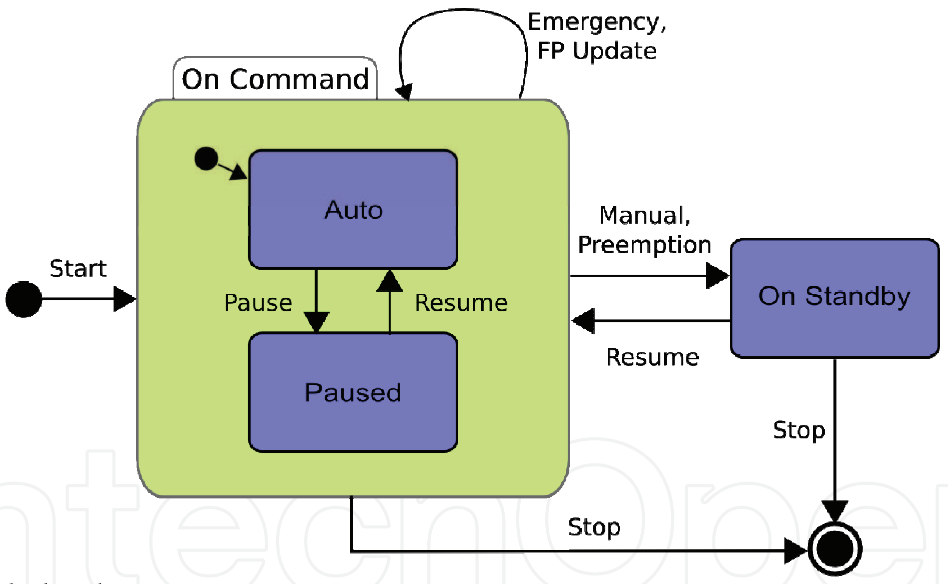


Figure 11. Flight Plan Execution Automata

Once the submitted flight plan has been loaded into the FPM, an internal representation is generated and the service is ready to start waypoint generation. The flight plan is represented by a tree whose root node corresponds to the whole flight plan (see Fig. 9). Stages are located at the next level of the tree, legs follow. At this point some degree of recursion can be found due to iterative legs, whose children legs form the body of the iterative structure. Finally each leg has an associated waypoint.

When the start command is received a traversal of the tree begins. The execution engine goes through each one of the flight plan stages, processing the legs they contain and generating waypoints as appropriate. Legs which present curved paths are approximated by sequences of waypoints.

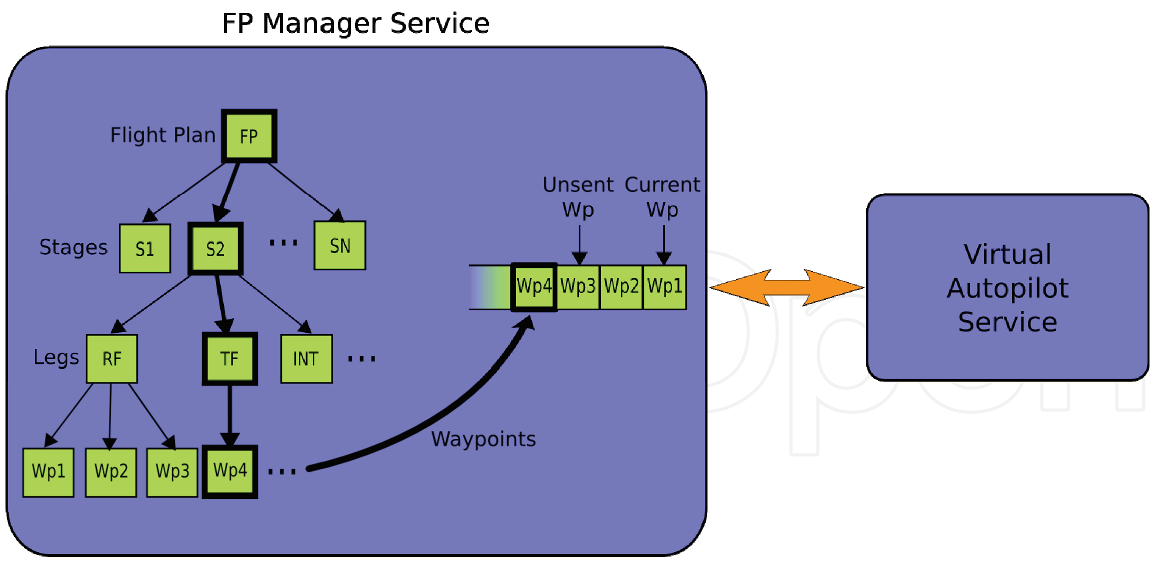


Figure 12. Overview of the Flight Plan execution procedure by the FPM

Waypoint generation works in a decoupled manner from the FPM operation depicted in Fig. 12. The FPM is implemented following a producer-consumer model. There is one worker thread in charge of waypoint generation. Each new waypoint is stored in a queue. The consumer will pass the waypoints from the queue on to the VAS. The head of the queue contains the waypoint the VAS is heading to. The queue is also used to keep track of unsent waypoints. Using this scheme, only the consumer needs to be aware of petitions or changes happening in the system (a waypoint has been reached, the flight plan updated, manual mode entered, etc.) and command the generation engine to take appropriate action if necessary (e.g. restart at a given leg). When a condition is encountered the producer will block and no waypoints will be generated until the condition outcome is available.
At this point the consumer will monitor the state of the queue and ensure that a decision is made in time, otherwise a default path, if present, will be taken. If the queue becomes empty (all waypoints have been flown), the VAS will be commanded to perform a holding pattern.

6. Conclusions

The class of mini/micro UAVs will become available to many research institutions, universities, and private companies to develop their research or commercial applications. However, current technology offers solutions for most components in the UAV system except systems to support and automate the actual sensing mission.
This chapter has introduced USAL, a service-oriented architecture designed to support the development of remote sensing applications. USAL offers a number of pre-defined services that can be easily parameterized to the specific needs of the application. Additional services can be introduced to incorporate new functionalities and at the same time reusing available services. A middleware designed to support the type of inter-service communications required by the USAL is also introduced.
A novel flight plan specification to be employed within the USAL has been also introduced. The proposed specification mechanism improves on the common list of waypoints approach by providing RNAV-like legs as the main unit for flight plan construction. Besides the leg concept is extended to include higher level control structures for specifying iterative

behaviour and branching. The decision making for this control structures can be based on mission variables, therefore enabling the UAV to respond depending on mission time information. The adaptability of the system to mission time circumstances is also greatly improved by the inclusion of parametric legs. With parametric legs, the flight path of the aircraft is dynamically generated depending on its input parameters.

7. Acknowledgments

This work has been partially funded by Ministry of Science and Education of Spain under contract CICYT TIN 2007-63927; and by the Innovative Studies Programme of the EUROCONTROL Experimental Centre, EUROCONTROL Innovative Studies: www.eurocontrol.int/eec/public/standard page/WP Innovative Studies.html.

8. References

W3C Working Group (2004), W3C Note 11: Web Services Architecture, *http://www.w3c.org/TR/ws-arch*, February 2004

UPnP Forum (2008), UPnP Device Architecture 1.0, *http://www.upnp.org/specs/arch*, April 2008

UAVNET Thematic Network (2005), European Civil Unmanned Air Vehicle Roadmap *http://www.uavnet.com*, March 2005

RTCA (2007), DO-304: Guidance Material and Considerations for Unmanned Aircraft Systems, March 2007

Santamaria, E.; Royo, P.; Lopez, J.; Barrado, C.; Pastor, E. & Prats, X. (2007). Increasing UAV capabilities through autopilot and flight plan abstraction, *Proceedings of the 26th Digital Avionics Systems Conference*, Dallas (TX), October 2007, AIAA/IEEE

Santamaria, E; Royo, P.; Barrado, C.; Pastor, E.; Lopez, J. & Prats, X. (2008). Mission Aware Flight Planning for Unmanned Aerial Systems, *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu (HI), August 2008, AIAA

Pastor, E.; Lopez, J. & Royo, P. (2007). UAV Payload and Mission Control Hardware/Software Architecture. *Aerospace and Electronic Systems Magazine*, Vol.22, No.6, June 2007 3-8

Lopez, J.; Royo, P.; Pastor, E.; Barrado, C. & Santamaria, E. (2007). A Middleware Architecture for Unmanned Aircraft Avionics, *Proceedings of 8th Int. Middleware Conference*, NewPort (CA), November 2007, ACM/IFIP/USEUNIX

Royo, P.; Lopez, J.; Pastor, E.; & Barrado, C. (2008). Service Abstraction Layer for UAV Flexible Application Development, *46th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, January 2008, AIAA

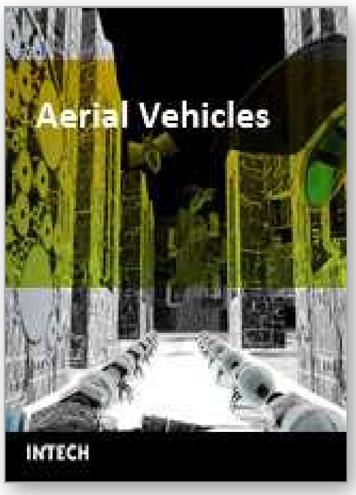
Cox, T.H.; Somers, I. & Fratello D.J. (2006). Earth Observations and the Role of UAVs: A Capabilities Assessment, 2006, NASA

Haiyang, C.; Yongcan, C. & YangQuan, C. (2007). Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey, *Proceedings of International Conference on Mechatronics and Automation (ICMA)*, Harbin, China, 2007, IEEE

FAA (2008). Aeronautical Information Manual, Official Guide to Basic Flight Information and ATC Procedures, *U.S. Federal Aviation Administration*, 2007.

EUROCONTROL (2003). Guidance Material for the Design of Terminal Procedures for Area Navigation, *European Organisation for the Safety of Air Navigation*

Schmidt, D.C. (2002). Middleware for Real-Time and Embedded Systems, *Communications of the ACM*, June 2002



Aerial Vehicles
Edited by Thanh Mung Lam

ISBN 978-953-7619-41-1
Hard cover, 320 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009