Escola Politècnica Superior de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE DE FI DE CARRERA

TÍTOL DEL PFC : Ground Control Segment for UAS Civil Applications

TITULACIÓ: Master in Science in Telecommunication Engineering & Management)

AUTOR: Raúl Cuadrado Santolaria

DIRECTOR: Pablo Royo Chic

DATA: November 4, 2011

Title : Ground Control Segment for UAS Civil Applications

Author: Raúl Cuadrado Santolaria

Director: Pablo Royo Chic

Date: November 4, 2011

Overview

The aeronautical industry is in a new era of growth thanks to numerous technological advances. One of them is the UAS; Unmanned Aircraft Systems are able to fly by themselves with autonomous operation capabilities.
UAS can advantage manned aircrafts in the so called *D-cube* applications: Dull, Dirty or Dangerous. Risks and costs can be minimized using UAS since they need no crew and their construction, operation and maintenance costs are comparatively lower.
This Master Thesis has been developed in the ICARUS (Intelligent Communications and Avionics for Robust Unmanned aerial Systems) research group, which is developing a new platform with UAS technology for civil use. It is true that this type of technology is already used in the military field for years, but it is now in the civil use where it can be expanded even more.
Under this scenario, a certain level of control and monitoring is required from the ground segment so that the pilot in command (PiC) is able to supervise the VAS operations. Therefore, it is needed an application in order to solve all pilot in command necessities. A real UAS flight has many different specific phases, for each one of the phases is needed a specific information to shown. This application has to control the UAS, the flight plans and will receive all the information that generates the UAS payload.
This Master Thesis solves some of the objectives of the ICARUS research group. One of the main parts of the project is the design and implementation of two applications for the Ground Control Station (GCS). In the GCS we need different workstations, each one for flight plan modifications. In GCS we need three different workstations, each one for different purpose. The first one is the workstation for the PiC to flight control (called Flight Monitor Service). The second one is the workstation for the PoC to manage the flight plan modifications (called Flight Plan Monitor Service) and the las one is for mission and payload management.
This project is focused in the design and development of the first two workstation.

*Quiero dar las grácias a mi familia
por ayudarme a tirar con todo hacia adelante y
a Alba por esperarme estos dos años.*

*Ante todo quiero dedicar este proyecto a Pablo
por aguantarme y ayudarme todo este tiempo.*

Grácias!!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ADS-B	Automatic Dependent Surveillance Broadcast
AI	Attitude Indicator
AMSL	Altitude above Mean Sea Level
AP	Autopilot
API	Application Program Interface
BLOS	Beyond Line Of Sight
CDTI	Cockpit Display of Traffic Information
CPU	Central Processing Units
FM	Flight Monitor
fpm	Feet per minute
FPMa	Flight Plan Manager
FPMo	Flight Plan Monitor
GCS	Ground Control Station
GIS	Geospatial Information Services
GPS	Global Positioning System
GUI	Graphical User Interface
HMI	Human Machine Interface
HTML	HyperText Markup Language
ICARUS	Intelligent Communications and Avionics for Robust Unmanned aerial Systems
IFR	Instrument Flight Rules
LAN	Local Area Network
LOS	Line of Sight
MAREA	Middleware Architecture for Remote Embedded Applications
MFC	Multi-function Display
MMo	Mission Monitor
PFD	Primary Flight Display
PiC	Pilot in Command
PoC	Pilot on Command
RAM	Random-access memory
SI	International System Units
SRS	Spatial Reference System
TCAS	Traffic Alert and Collision Avoidance System
UAS	Unmanned Aircraft Systems
UAV	Unmanned Aerial Vehicle
UI	User Interface
USAL	UAS Service Abstraction Layer
VAS	Virtual Autopilot Service
VLOS	Visual Line of Sight
VFR	Visual Flight Rules
WMS	Web Map Service
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

INTRODUCCTION

The Unnamed Aircraft Systems (UAS) are aircraft that can fly without a pilot; that is, an airframe and a computer system which combines sensors, a Global Positioning System (GPS), servos and Central Processing Units (CPUs); all these elements combined have to pilot the plane with no human intervention. UAS can be remote controlled or fly autonomously based on pre-programmed flight plans. The UAS, for years, have been used only for military applications. Today the UAS industry is growing because the possibility to use it in expensive and dangerous civil applications. UAS can advantage manned aircraft in their ability to perform so called 3D jobs: Dull, Dirty or Dangerous. Risks and costs can be minimized using UAS since they need no crew and their construction, operation and maintenance costs are comparatively lower.

One of the main differences between a UAS and a manned aircraft is in the distribution of the UAS. Manned aircraft elements are all contained within the aircraft. A UAS comprises system elements in three major segments:

- Air Segment: this segment includes one or more Unmanned Aircrafts (UA) with their payloads. Each of the UA includes the airframe and the avionics and propulsion system. The payload consists of systems that support the intended mission capabilities. It is made up of the required systems for the mission such as cameras, sensors, antennas, warfare equipment,etc.
- Communications Segment: this is divided into the Command & Control data link, the Payload data link and External Communications. The main link categories are defined according to the distance at which the UAS is operating: Visual Line of Sight (VLOS), Line of Sight (LOS) and Beyond Line Of Sight (BLOS).
- Ground Segment: distributed in different parts, the Ground Control Station (GCS), the Payload Control Station/ Ground Data Terminal (GDT) and if necessary the Launch and Recovery System (LRS). In the ground segment, the GCS is the most important part. It includes all the required equipment for the UA pilot, flight planning and mission monitoring. Also, it translates pilot inputs into the appropriate commands to be transmitted over the communication link to the aircraft segment.

The ICARUS research group proposes a GCS designed to support any kind of mission. The GCS has three different workstations (see Figure 1), each one for a different users profile; The Flight Monitor Workstation (FM), the Flight Plan Monitor Workstation (FPMo) and the Mission Monitor Workstation (MMo).

- Flight Monitor Workstation: The FM is designed for the pilot in command (PiC) users profile. This workstation monitors all the telemetry, electrical, engine and alarms information of the UAS. The PiC has to be able to control the UAS from this workstation with all the necessary information to do it. The user does not only need visual information, also needs a joystick with throttle to manually control the UA. The FM workstation has to be a standalone station, that is it has to control the UAS flight without the other two workstations.

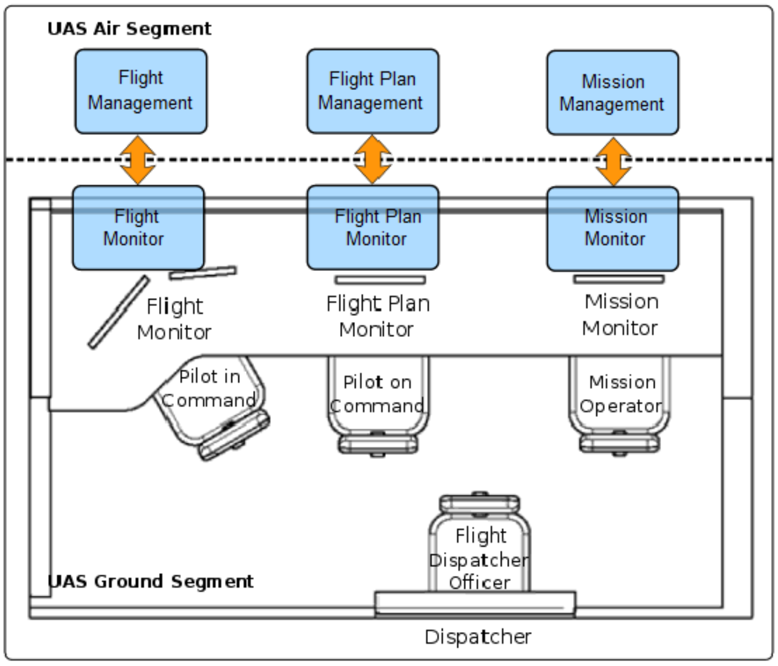


Figure 1: Ground control station distribution.

- Flight Plan Monitor Workstation: The FPMo is designed for the pilot on command (PoC) users profile. The Flight Plan Manager (FPMa) capabilities are managed with the FPMo; by means of this workstation we can monitor and edit the flight plan. The capabilities required of the FPMo are related to inherent dynamic behaviors offered by the FPM and the surrounding services that help manage in-flight contingencies, takeoff and landing operations, etc

- Mission Monitor Workstation: During a real flight mission this workstation will be the most important one. The MMo is the workstation used to manage the mission and the payload. All the information produced by the UAS payload as the sensors or cameras will be shown in this workstation for the post process.

This thesis focuses on the development of two of the workstations of the GCS, particularly the FPMo and FM. These stations can be analyzed as a set of elements that provide information on the status of UAS or elements that allow interaction with it.

In order to reuse the work done for these two workstations, it was decided to implement the stations from small modules or controls. For this reason one of the main components developed for this project is a library of controls which contains all the little pieces that make up the FPM and the FM.

Objectives

The main objective of this project is to develop the Flight Monitor (FM) and Flight Plan Monitor(FPMo) workstations for the ICARUS Ground Control Station. Inside this big main objectives, four different goals were fixed at the beginning of the thesis:

- The workstations have to be stable. The FM and FPMo do not have to implement a lot of features but they have to be robust to serious failures (execution can not be interrupted).
- The stations have to be implemented in small controls. This controls are grouped in a library of controls in order to be reused.
- Both workstations need control to visualize data over a map. For this reason it is necessary to develop a control for visualizing this data without Internet connection.
- The final application has to be integrated inside the ICARUS simulator (called ISIS). It has to take into account the platform specifications, the requirements and the interaction with other simulator components.

CHAPTER 1. BACKGROUND

1.1. Middleware Architecture for Remote Embedded Applications (MAREA)

For implementing UAS civil missions we use a distributed embedded system that will be on board of the aircraft and that will operate as a payload/mission controller. 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. 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 UAS mission.

MAREA is a middleware-based software system used to communicate different services over the local area network (LAN)[3]. MAREA provides an execution environment with communication channels and common functionalities. The role of each service is expressed by the action of publish, subscribe, or both simultaneously; in this way, the publish/subscribe model eliminates complex network programming for distributed applications that makes it easy to implement an embedded service. MAREA offers the localization of the other services and manages their discovery within the network; it handles all the transfer chores, message addressing and retransmission,data delivery, flow control, etc.

MAREA promotes a publish/subscribe model for sending and receiving data, events and commands among the services of the UAS. Services that produce valuable data publish this information while other services may subscribe to them. MAREA is responsible for delivering the information to all subscribers that declare an interest in the topic. Information exchange is carried out through four communication primitives, which have been named as Variables, Events, Remote Invocations and File Transmissions.

- Variables. Variables are the transmission of structured, and generally short, information from a service to one or more services of the distributed system.
- Events. Like Variables, Events also follow the publication-subscription paradigm. The utility of Events is to inform of punctual and important facts to all the services that care about.
- Remote Invocation. The Remote Invocation is an intuitive way to model one-to-one of interactions between services.
- File Transmission. The File Transmission primitive is used basically to transfer long file-structured information from a node to another when exists the need to transfer continuous media with safety variables. Variables are the transmission of structured, and generally short, information from a service to one or more services of the distributed system.

1.2. USAL: UAS service abstraction layer

Providing a common infrastructure for communicating isolated UAS 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. From the study and definition of several UAS missions, one can identify the most common requirements and functionalities that are present among them.

The UAS Service Abstraction Layer (USAL) is the set of available services running on top of the UAS architecture to give support to most types of UAS missions [1]. USAL can be compared to an operating system. Computers have hardware devices used for input/output operations. Every device has its own particularities and the OS offers an abstraction layer to access such devices in a uniform way. Basically, it publishes an Application Program Interface (API) which provides end-users with efficient and secure access to all hardware elements. The USAL considers sensors and in general all payload as hardware devices of a computer. The USAL is a software abstraction layer that gives facilities to end-users programs to access the UAS payload. The USAL also provides many other useful features designed to simplify the complexity of developing the UAS application.

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 every mission. Only those services required for a given configuration/mission should be present and/or activated in the UAS.

Available USAL services have been classified in three categories:

- Flight services: all services in charge of basic UAS flight operations:autopilot, basic monitoring, contingency management, etc.
- Mission services: all services in charge of developing the actual UAS mission, controlling the payload and the area of surveillance, processing or saving the earth observation information and showing it to the end users.
- Payload services: specialized services interfacing with the input/output capabilities provided by the actual payload carried by the UAS.

1.3. Flight Plan Specification Language

In order to manage unmanned vehicle in autonomous mode it is necessary to load a flight plan. The flight plan is a document that specifies the route the aircraft will follow. Common UAS autopilot systems use as a mechanism for flight plan execution a list of points. However this method has some limitations to specify a complex mission. For that reason in the ICARUS group we have been developed a complex flight plan language [2]. The flight plans defined using waypoints have several important limitations:

- It is difficult to specify complex trajectories and it does not support constructs such as forks or iterations.
- It is not flexible because small changes may imply having to deal with a considerable amount of waypoints
- It is unable to easly adapt to mission circumstances.
- It lacks constructs for grouping and reusing flight plan fragments.

To improve current UAS operation a flight plan language with richer semantics is proposed, which enables flight progress to adapt to mission circumstances. The flight plan is stored in an XML document that will be submitted to the UAS in order to carry out its execution.

A flight plan specifies the path followed by the aircraft using several items. As seen in Figure1.1, each flight plan is composed of a sequence of stages, such as take-off, departure procedure and others. Each flight plan stage is composed of a structured collection of legs. The leg is used to specify the trajectory followed by the aircraft to reach a given waypoint from the preceding one. In the simplest case this trajectory will be a straight line.

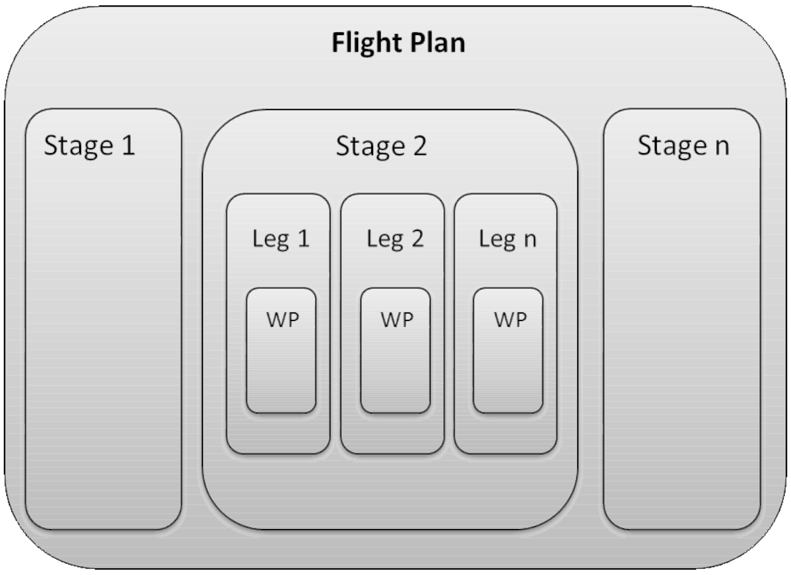


Figure 1.1: Main flight plan structure

CHAPTER 2. GEOGRAPHIC INFORMATION SYSTEM MANAGEMENT

This chapter describes the component implemented to visualize georeferenced information. This component allows map visualization from different providers, and representation of georeferenced data, such as UAS path, flight plan, etc. The first objective of this component is to provide the Flight Plan Monitor Workstation an optimal way for illustrating the UAS path and UAS flight plan on a map.

Previous ICARUS applications [4] use the ShapMap library [7] to seize the opportunities offered by Geospatial Information Services (GIS) to provide map data. SharpMap is a mapping library to use in web and desktop applications. It provides access to many types of GIS data, enables spatial querying of that data, and render maps.

The main problem of this library is that it can not save maps locally. Therefore Internet connection is necessary to load all maps. UAS usually involve work in areas where it is difficult to access. In many cases the GCS is located in places where it is impossible to have Internet access so it is impossible to load the maps. There are other solutions like the paid version of Google Earth Portable [15], but the price of the product, for the performance required, is too high.

For solving this problem we have decided to use the BruTile library [6], which is able to provide a map data and to save it a local cache. The main BruTile problem is that it can be used to display maps but it does not allow to represent any object on the map. Therefore, the second project decision was to develop a mechanism to link a georeferenced data representation with BruTile data.

2.1. Specifications

The minimum requirements of this software component are described below:

- Georeferenced 2D map display.
- Control operation without Internet.
- Loading maps using WMS [14] standard.
- Loading Google Maps maps [16].
- Georeferenced objects on map.
- Map movement by mouse.
- Zoom in and Zoom out.
- Zooming in a given area.
- Centering the map on a point.

Another important requirement is that it can be reused. To accomplish this requirements we decided to implement a fully independent control to GCS applications. The control has been implemented for reusing in other applications minimizing the effort to adapt the applications.

2.2. Architecture

The manager of Geographic Information is called UcMapControl. It is a visual control which has been programmed using .Net. To develop the control core C# has been used, and the presentation layer is implemented using WPF [10]. This control is divided into two blocks (Figure 2.1), one block is the BruTile map control and the other block contains the layers to show georeferenced data.

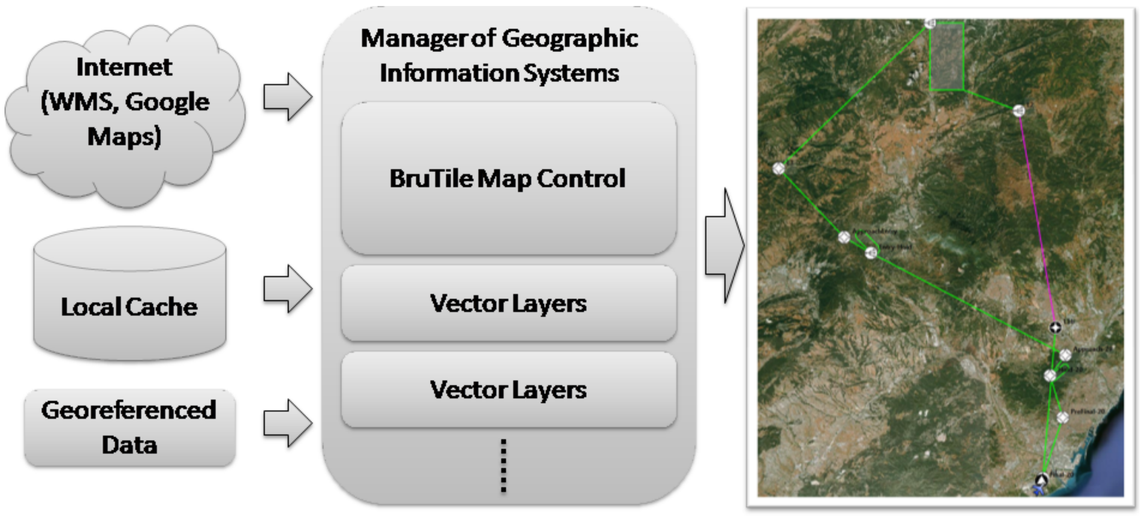


Figure 2.1: Geographic Information System Management architecture

The BruTile map block is responsible visualizing GIS data obtained from WMS or GoogleEarth servers. It manages the GIS data cache and provides the main mechanism for interaction between the user and the map, such as moving the map or the zoom switch.

In order to show different georeferenced data we have designed a mechanism based in layers. The layer is the element that allows drawing referenced items on the map. Also it provides the mechanisms needed to transform particular data in graphical form. Each layer is oriented to solve a particular problem. Several layers have been developed for this project:

- Layer to display flight plans.
- Layer to display the UAS path.
- Layer to display landmarks.
- Layer to display a runway.

- Layer to visualize the UAS waypoint destination.

2.3. Design

In this section we are going to describe the different elements and resources that make up the UcMapControl control. Special attention is paid to the basic components that have been implemented, explaining the use and the basic attributes that compose them.

Figure 2.2 shows the design diagram of the control. The Manager of Geographic Information, called UcMapControl, is the basic component. UcMapControl contains all other controls. It is responsible to give the map source to BruTile Mapping Control and manages the user interaction. Also the UcMapControl synchronizes the vector layers. The UcMapControl provides the map source to BruTile Mapping Control [8] . The GIS provider manager delivers to UcMapControl the server configurations. Finally, Spatial Reference System (SRS) manager is the object responsible to do the geographic point conversion between vector layers and BruTile control.

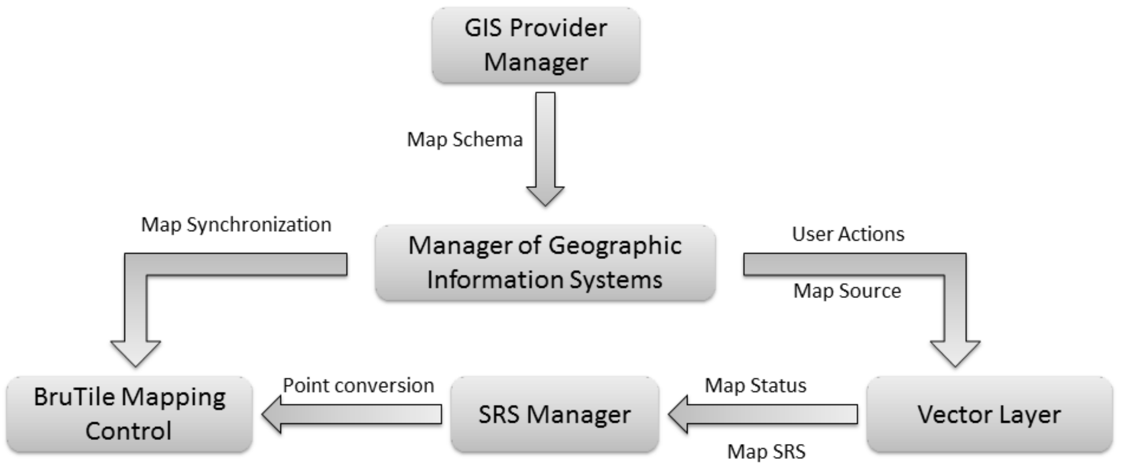


Figure 2.2: Geographic Information System Management design

These components are described in more detail in following sections.

2.3.1. BruTile Mapping Control

In order to manage the map data we have used a BruTile Mapping Control called MapControl. The BruTile control separates drawing from data fetching. The renderer only render the data it has available in memory at that instant. The fetcher runs on background thread and receives messages from the UI thread telling it what to fetch.

The Figure 2.3 shows how BruTile Mapping Control data fetcher works. The UI and Fetcher communicate through non blocking messages. Whenever the user pans or zooms, a View

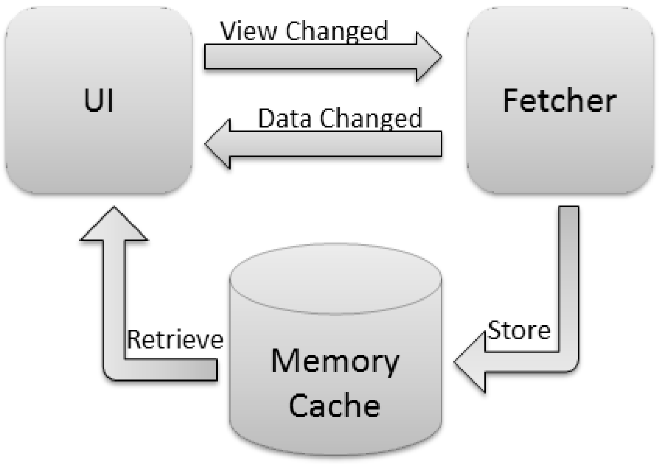


Figure 2.3: BruTile mapping control design

Changed message is sent to the Fetcher. Whenever new data arrives a Data Changed message is sent to the UI. The fetcher dumps incoming data into a cache. The UI renderer retrieves whatever is needed from that cache when rendering.

Both the fetcher and the renderer can use all kinds of smart tricks. In case of tiling, the fetcher can pre-fetch tiles based on its current view, or on the way the view changes over time. The renderer could search for alternative tiles (higher or lower levels) when the optimal tiles are not available. Those strategies should be tuned to support each other. For instance, in the current implementation the renderer uses high level tiles when the optimal tiles are not available, and the fetcher pre-fetches high level tiles to assist the renderer. But the way they play together is not specified in the interface. This loose coupling keeps things simple and flexible and the renderer has never to wait for the fetcher. This results in a good (perceived) performance.

All events, launched by the user interface, are redirected to UcMapControl. UcMapControl has the control of BruTile map and it can synchronize the vector layers. The fetcher and data collection is managed by BruTile library, however UI management is doing by SharpMap core. To specify the source of the data map is doing by iTileLayer class and which is a composed to a schema (iTileSchema) and a provider (iTileProvider).

The iTileSchema has the function of defining the map schema that will be applied to the server. The iTileSchema does not provide all the necessary features so that later we can work with the map coordinates. We have decided implement another class that extends iTileSchema, called ExtendedISchema. It specifies, among other things, the size of each tile, the coordinate system to requests the map data and more.

2.3.2. Vector Layer Manager

In order to draw georeferenced data on the map we have implemented a component called MapControlLayer. Following a pattern we can design a specific layer for each application. The design of each layer aims at displaying a particular type of data in each layer (waypoints, flight plan, landmarks). MapControlLayer class is the interface that provides the necessary methods to synchronize each layer with the displayed map.

To develop Flight Plan Monitor and Flight Monitor workstations, we have implemented several layers:

- UAS Layer: This layer displays the path of the UAS by means of the heading and the current position.
- Runway Layer: This layer displays a specific runway
- Reference Point Layer: This layer displays reference points in a map from decimal degrees coordinates.
- Destination Layer: It draws a straight line from the current position of the UAS to a waypoint.
- Flight Plan Layer: This layer illustrates the flight plan over the map. In addition this layer implements the necessary methods for Leg [1] selection and Leg configurations. Due to the complexity of this layer, it has been implemented using several objects (see Figure 2.4).

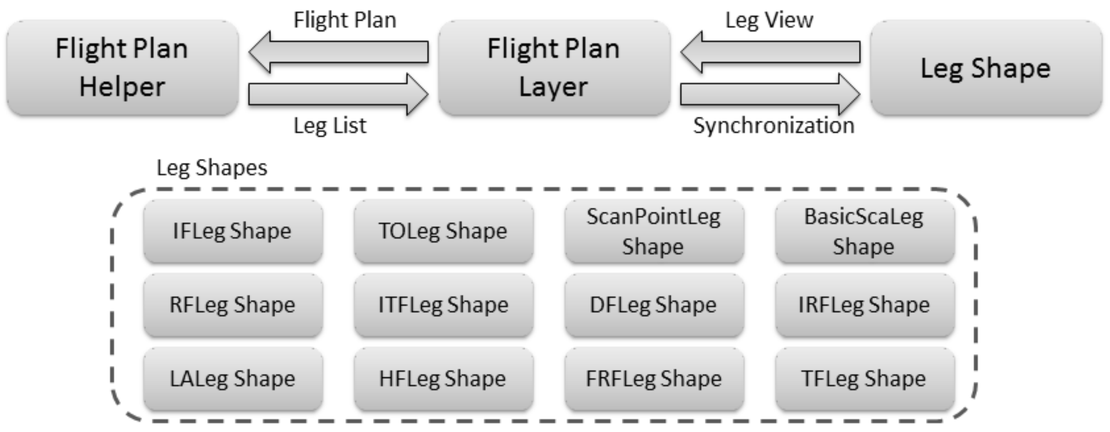


Figure 2.4: Design of fligh plan layer

This layer is composed by three software components: Flight Plan Layer, Leg Shape and Flight Plan Helper.

- Flight Plan Layer manages the user interaction and the shapes generation.
- Flight Plan Helper is responsible of listing the legs that compose a flight plan or the legs of the path selected in the flight plan.
- Leg Shape is the main class to draw a Leg. It implements the methods needed to update the view of the Leg and looks for the start and destination point to

[1]A leg specifies the fight path to get to a given waypoint. In general, legs contain a destination waypoint and a reference to the next leg.[2]

draw it. It also has the methods needed to change any of the display Leg parameters (leg color selected, unselected leg color, visibility). From the main class LegShape, we have implemented a new class for each leg, such as BasicScanShape, DFLegShape, etc.

2.3.3. Spatial Reference Systems Manager

Spatial Reference Systems manager makes possible to use different spatial reference systems between layers and map. This component uses a reference system from the BruTile map control source It allows all layers to be added and georeferenced properly. The SRS manager contains different methods which enable to convert the data points from radians or degrees to screen points.

2.3.4. GIS Providers Manager

This module is used for generating the necessary object to configure the BruTile map control from a XML[2] (Extensible Markup Language) file. This component makes ITileSource object for WMS and WMS-C servers.
The map can be set to allow WMS or Google Maps servers provider. This object do not require any configuration if you want to load data from Google server. In case you want to load maps from a WMS server, an XML file needs to be generated because each server has a different configuration

2.4. Implementation

This section explains the implemented classes and the methods to connect all software components. Figure 2.5 shows the class diagram of the UcMapcontrol and the main classes that compose it. The control is composed of four main areas, the ExtendedSchema, the MapControl instance, the MapCoordinates and finally a list of MapControlLayers. The ExtendedSchema class will not be described in the following sections because it is only an object that contains information. It gives the map provider information in order to do the conversion using MapCoordinates class.

2.4.1. UcMapControl

This control is composed of a several files. This section only shows the methods to use the UcMapControl. The control is implemented as a UserControl subclass. A UserControl is a content control, which means that it can contain a single object of any type (such as a string, an image, or a panel). Deriving from UserControl is a suitable model if you want to build a control by adding existing elements to it, similar to how you building an application.

[2]Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879).

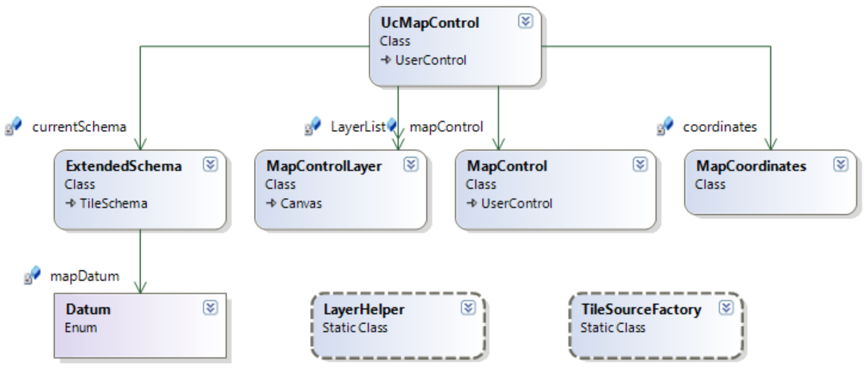


Figure 2.5: UcMapControl class diagram

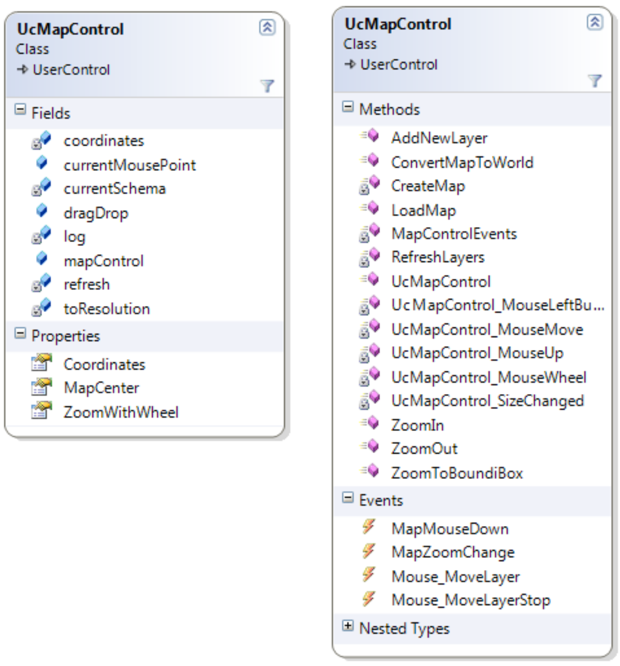


Figure 2.6: UcMapControl class diagram

The control is composed of two subcontrols, a Grid and MapControl. The Grid is the object that provides space to host the BruTile control and vector layers. The map control has been described in Section 2.3.1.

The Figure 2.6 shows the class diagram of the control. In it we see the different classes and methods implemented in UcMapControl. In this figure we have loaded a map using Google Maps and we have added two layers. One layer shows the position of UAS and its path, and the other shows the loaded flight plan.

The vector layers are stored in a list in order to synchronize them with the map control. With the purpose of better synchronizing the vector layers and map, we have decided to disable the BruTile map control user events. The UcMapControl catches this event and throws the appropriate method in MapControl. By catching the events we know what is happening in the control. The methods and the events implemented in this control are shown in Table 2.1.

Type	**Name**	**Description**
Method	LoadMap	(ITileSource source) Loads a new map
Method	AddNewLayer	Loads a new vector layer in the control
Method	ConvertMapToWorld	Returns the coordinates of a screen point
Method	ZoomIn	Decreases by 2 the visible map area
Method	ZoomOut	Increases by 2 the visible map area
Method	ZoomToBoundiBox	Zooms to a specific area
Method	ZoomWithWheel	Enables or disables the option to zoom with the mouse.
Method	MapCenter	Get or set the current map center
Event	Refresh	This event is executed when it is necessary to refresh layer contents.
Event	MapChangeZoom	This event is throw when map zoom is changed
Event	MapMouse_Down	When user click with mouse right button this event is
Event	Mouse_MoveLayer	This event is thrown while the map is moving
Event	Mouse_MoveLayerStart	This event is thrown when the map is beginning to move
Event	Mouse_MoveLayerStop	This event is thrown when the mouse button returns to idle state

Table 2.1: UcMapControl methods and events

2.4.2. Vector Layers

The class to define the vector layer behavior is called MapControlLayer. It is the parent class of all vector layers to be implemented. The MapControlLayer is a container for WPF controls implemented using a Canvas panel.

The Canvas is the most basic layout panel in WPF. It is child elements are positioned by explicit coordinates. The coordinates can be specified relative to any side of the panel using the *Canvas.Left, Canvas.Top, Canvas.Bottom* and *Canvas.Right* attached properties.The panel is typically used to group 2D graphic elements together and not to layout user interface elements. This is important because specifying absolute coordinates brings you in trouble when you begin to resize, scale or localize your application.

Vector layers are controlled by UcMapControl. They implement the methods needed to synchronize the content with the data displayed by the MapControl BruTile. Figure 2.7 shows the implementation of this class. The fields that implement this layer have been explained in previous sections. View class contains information on the position and zoom level of the MapControl. This information is used for a preview before upgrading the components of the layer.

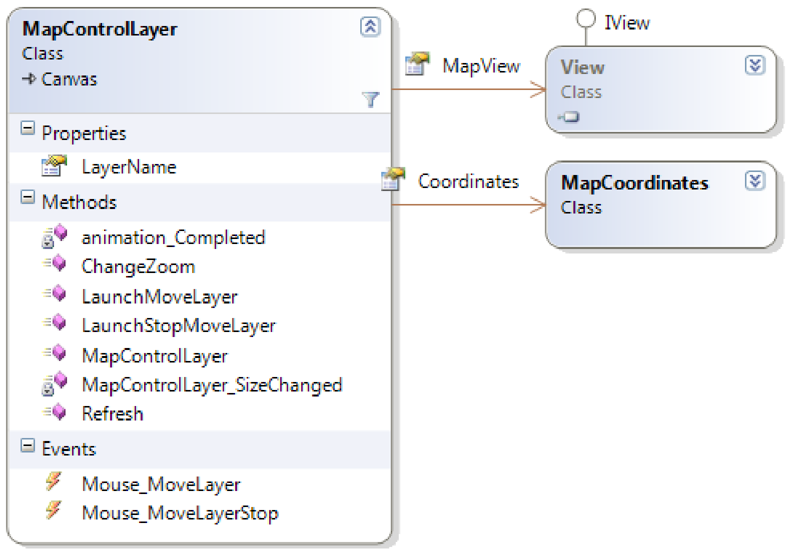


Figure 2.7: MapControlLayer class diagram

In order to deploy a new layer, the developer has only to implement the Refresh(), LaunchStopMoveLayer() and LaunchMoveLayer() methods because the zooming and resizing methods are already implemented. These methods generate an approximate view after the Refresh() method properly updates the content. The Refresh() method is responsible for recalculating all the coordinates of the objects added to the layer. It turns them into pixels and it recalculated the points, once instead of updating every time when you move the mouse. As mentioned in Section 2.3.2., we needed to implement different vector layers to achieve the system requirements. The vector layers implementations are described below.

- UAVMapControlLayer: This layer is the implementation of UAS vector layer. The UAVMapControlLayer displays the position and path followed by the UAS. This class consists of an image and an object of Path type.The Path class enables to draw curves and complex shapes. These curves and shapes are described using Geometry objects. This class draws the path of the UAS. In order to improve performance when updating the layer, we do not paint all the points received from the UAS. According to the zoom level, at the moment of updating the layer, the positions are only painted on the screen when belong to a different pixel. When the zoom of the map is small (the map looks away) many positions are superimposed. Thus calling the Refresh method reduces the processing. To indicate the current position of UAS we use an image of a plane. This image is positioned on the screen depending on the UAS latitude, longitude and heading.

- RunWayMapControlLayer: The RunWayMapControlLayer is the implementation of runway vector layer. This layer displays a runway from the pre-configured definition. This layer generates a path object connecting the specified pre-configured points.

- ReferenceMapControlLayer: This layer is used to place an Image object in a user-specified point. The Image object has a default icon and default size.

- DestinationMapControlLayer: The DestinationMapControlLayer is the implementation of destination vector layer. This layer consists of displaying a line between two specified points. It first adds the destination, and then, updates the initial position with the UAS position. The line is drawn using the object of type Line. The destination is obtained from the UAS telemetry.

- FlightPlanMapControlLayer: This layer is the implementation of flight plan vector layer. The FlightPlanMapControlLayer is made up of a several objects displayed on the screen. Unlike the other layers, the FlightPlanMapControlLayer does not provide mechanisms for updating the content. In this layer to improve performance and increase scalability we have decided to create different objects with individual synchronization. In order to illustrate the Legs as a drawing we have implemented a LegShape class. The diagram class of this layer and the LegShape implemented are shown in Figure 2.8.

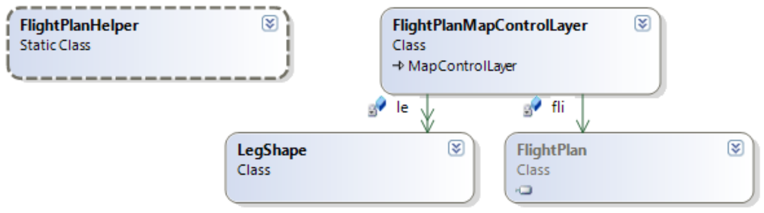


Figure 2.8: FlightPlanMapControlLayer class diagram

The LegShape is responsible for searching the start points and end points of a Leg. A Leg is defined by a destination point, so to draw a path we need to know the destination point of the previous Leg. When the flight plan is composed by Intersection and Iterative Legs we must take into account a number of rules. For this reason it has been developed an algorithm to obtain the initial point or points.

The LegShape class also identifies the waypoint type with different icons depending on whether the IntersectionLeg is a fly-by or fly-over. From LegShape class, other classes only have to implement the methods needed to transform the definition of each Leg in its corresponding drawing.

2.4.3. MapCoordinate

MapCoordinate implements the methods to perform coordinate conversions. This class converts decimal degrees (DecDeg) or radians coordinates into the coordinate system used by the map. To perform this function the MapCoordinates use the map source schema and the current view of the map (see Figure 2.9).

The source schema is implemented in ExtendedSchema, it provides information about spatial reference system used to request the map data. The View object provides information about the current status of BruTile map control. This class provides information about

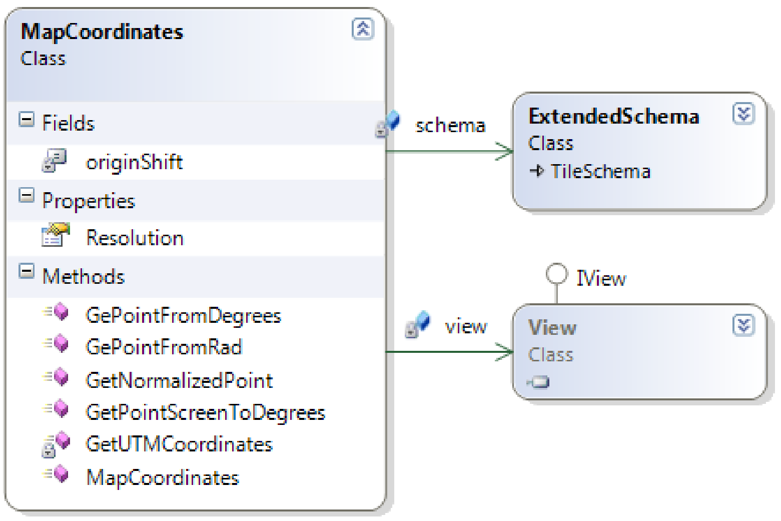


Figure 2.9: MapCoordinates implementation

the displayed map area and the applied zoom. The Table 2.2 shows the methods to do the conversion.

Type	**Name**	**Description**
Method	GePointFromDegrees	Converts from point in degrees to point in the reference system used by the map loaded into the control.
Method	GePointFromRad	Converts from point in radians to point in the reference system used by the map loaded into the control.
Method	GetNormalizedPoint	Get the screen location of a point.
Method	GetPointScreenToDeg	Converts from point in pixels to point in degrees.

Table 2.2: MapCoordinate methods to do the conversions

CHAPTER 3. GRAPHICAL CONTROLS LIBRARY FOR THE GROUND CONTROL STATIONS

In this section we are going to explain the different controls needed to develop the Flight Monitor and Flight Plan Monitors workstations. The workstations have required many elements to manage UAS operations. In order to reuse these elements we have created a controls library where we added all developed controls. Thus, each control can be reused in different workstations.

The software controls that have been developed in our library are:

- Primary Flight Display
- Cockpit Display of Traffic Information
- Manager of virtual autopilot states.
- Video and audio player and grabber.
- Interface for Joystick Management
- Google Earth Synthetic View

3.1. Primary Flight Display

A primary flight display or PFD is a modern aircraft instrument dedicated to flight information. Representations of older aircraft instruments are combined on one compact display, simplifying pilot work flow and streamlining cockpit layouts.The details of the display layout on a primary flight display can vary enormously, depending on the aircraft and the aircraft's manufacturer. However the great majority of PFDs follow a similar layout convention.

The center of the PFD contains an Attitude Indicator (AI), which gives the pilot information about the aircraft's pitch and roll characteristics, and the orientation of the aircraft on the horizon. On left and right of the attitude indicator there are usually the airspeed and altitude indicators, respectively. The airspeed indicator displays the speed of the aircraft in knots, while the altitude indicator displays the aircraft's altitude above Mean Sea Level (AMSL). Both of these indicators are usually presented as vertical tapes, which scroll up and down as altitude and airspeed change.

The vertical speed indicator, usually located next to the altitude indicator, indicates how fast the aircraft is ascending or descending, or the rate at which the altitude changes. This is usually represented with numbers in thousands of feet per minute. For example, a measurement of +2 indicates an ascent of 2000 feet per minute (fpm), while a measurement of -1.5 indicates a descent of 1500 feet per minute. Usually the PFD also contains the heading display. The heading control has been implemented separately from the control because it will be reused for different Graphical User Interface (GUI) components.

3.1.1. Design and Implementation

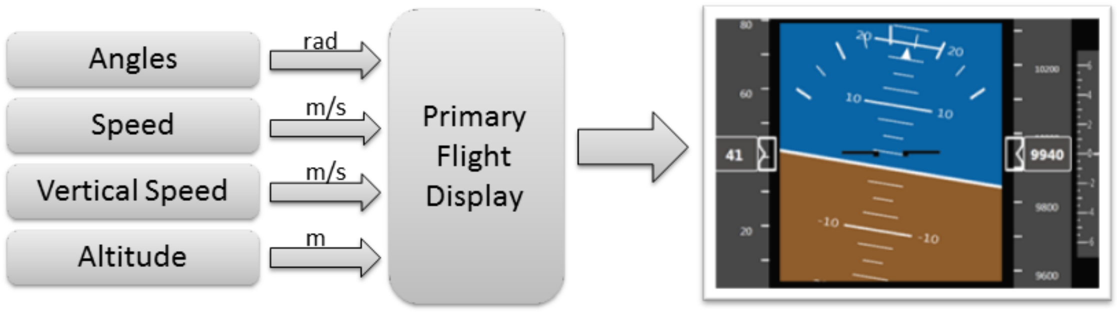


Figure 3.1: Architecture of Primary Flight Display control

The operation of this control is very simple. As we see, this control transforms telemetry data into a visual representation. The input data is compatible with the International System Units (SI), so it becomes in values used by the cockpit instruments. The visual layer of the control consists of a set of images (see Figure 3.1). This image are moved across the screen depending on the input values. We have decided to load some image instead of painting the visualization through code because it generates lees overhead on the CPU.

The control layout has been done in a way that each indicator of the PFD is independent. We have defined different methods to update the values. The list of control methods is described in Table 3.1.

Type	**Name**	**Description**
Method	UpdatePitchAndRoll	Update the UAS pitch and roll
Method	UpdateAltitude	Update the UAS altitude
Method	UpdateVerticalSpeed	Update the UAS vertical speed

Table 3.1: Methods to update Primary Flight Display control

3.2. Cockpit Display of Traffic Information

Cockpit Display of Traffic Information (CDTI) has the function of presenting surveillance information about the surrounding traffic to the flight crew. The information presented includes the relative position of other aircraft in the vicinity with respect to the aircraft itself. Traffic information for the CDTI is obtained from automatic dependent surveillance broadcast(ADS-B)

The operational goals of the CDTI are to improve the safety and efficiency of flight operations through an enhanced surveillance capability coupled with new operational procedures. Early applications of CDTI included the following objectives and applications.

- Improve traffic situational awareness
- Improve the probability of visual acquisition of traffic compared with unaided visual search
- Allow aircraft to obtain more efficient altitudes and tracks in oceanic/remote airspace.

3.2.1. Design and Implementation

The implemented CDTI control is called UcCDTI. As seen in Figure 3.2 UcCDTI control consists of two parts. One part shows the UAS heading. The other one shows the traffic in front of the UAS. This control has several input parameters. On one hand it requires the UAS telemetry (heading, position and speed). And on the other hand the ADS-B messages, which allow to update the traffic aircraft position on the screen. In order to manage the aircraft independently we have implemented AirCraft class. The AirCraft object is responsible of the orientation and the refreshment of displayed aircraft.

We have different methods to refresh the control and update its content. The list of control methods is described in Table 3.2.

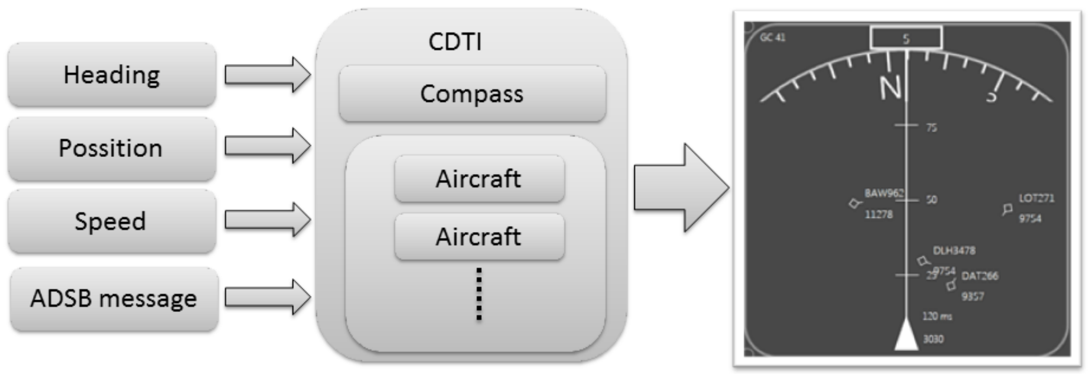


Figure 3.2: Cockpit Display of Traffic Information control architecture

Type	Name	Description
Method	NewADSBMessage	Add new ADSB message
Method	UpdateTelemetry	Update the position, heading and speed of our UAS

Table 3.2: Methods to add new ADSB message and update UAS telemetry

3.3. Manager of Virtual Autopilot States

The Virtual Autopilot Service is the component that interacts between the autopilot and the rest of the components of the USAL. It is a service that provides a standardized interface to the particular autopilot on board. VAS is responsible for managing the autopilot states.

The aim of the VAS states is to increase the autopilot functionalities and to give the same services without depending on the on-board autopilot. There are a large number of states and a large number of possible transitions. Due to this complexity the PiC needs a mechanism to change VAS state safely. The objective of this control is to provide intuitive support for changes between states.

3.3.1. Design and Implementation

The change of state on the VAS requires a simple control that shows the allowed state transitions, and then displays the current status.

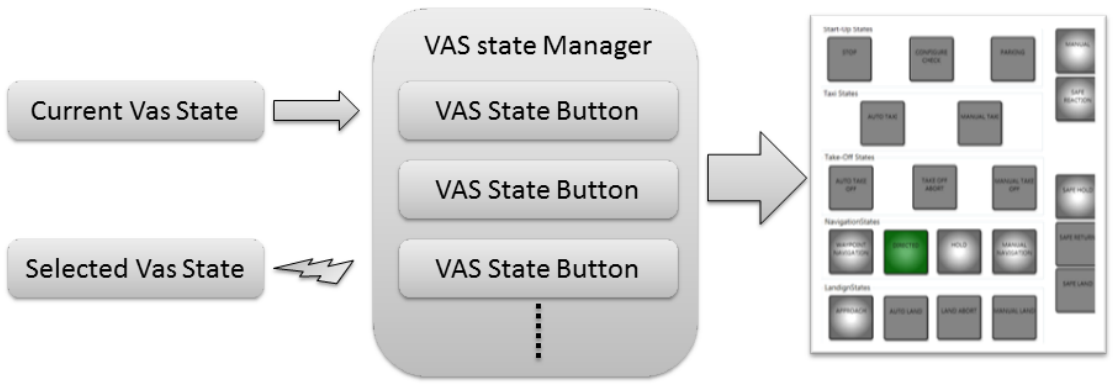


Figure 3.3: Diagram of Virtual Autopilot states control

As shown in Figure 3.3 the VAS state manager (called UcVasScreen), has one output and one input. The operation of the controls is as follows. First, the current VAS state, is introduced at the control and next, the control updates the buttons and enables the corresponding buttons. In the next step the user selects the transition and the control notifies the new state to the VAS.

The UcVasScreen acts as a container of several objects called UcVasStateButton. This object operates as a button and manages its operations to allow to request the state change. The UcVasStateButton have several state which are indicated by its color.

- Dark Gray - The state is disabled and it is not possible. The transition to this state is wrong.
- Gray - The state is disabled and it is not possible the transition of this state.

- Blue - The user has click the button once time. If the button is pressed again the buttons turns gray. If after the user have pressed the button in blue state and VAS does not change the state during five seconds, the button turns gray again.
- Green - In this color the VAS has been replied with a state changed. This color indicates the current state.

The main advantage of having the state management in the button is that it is easy to adapt VasState control, if new states are added or removed.

The list of control methods is described in Table 3.3.

Type	**Name**	**Description**
Method	OnVasStateChange	This method update the current VAS state at control
Event	ChangeCurrentVasState	The control throws this event to request the state change

Table 3.3: Input and Output of VAS states control

3.4. Display and Graph Generator

The Display and Graph Generator aims at showing the hierarchical flight plan structure as a graph. As seen in section 6 the structure of a flight plan can be very complex. In simple flight plans unique path is found, it can be enough to display a flight plan on the map. When we are managing a flight plan with forks and iterative legs, the map view becomes very confusing. Therefore we need another kind of flight plan representation when alternative or iterative Legs are present. Therefore, the Display and Graph Generator generates a flight plan view as a tree.

3.4.1. Design and Implementation

Figure 3.4 illustrates the Display and Graph Generator architecture. The name of this control is UcGraphControl. The UcGraphControl is a container of nodes and their relations. Each node is implemented through UcGroupNode and UcNode objects. The GroupNode allows group some Nodes.

The main function of uCGraphControl is to generate the nodes and their relationships from a flight plan. The Graph class is the class that owns the primitives needed to generate the graph. This class understands only nodes and relations between them. From this information generates the Nodes list that will be displayed and their positions.

Table 3.4 show the methods to interact with this control.

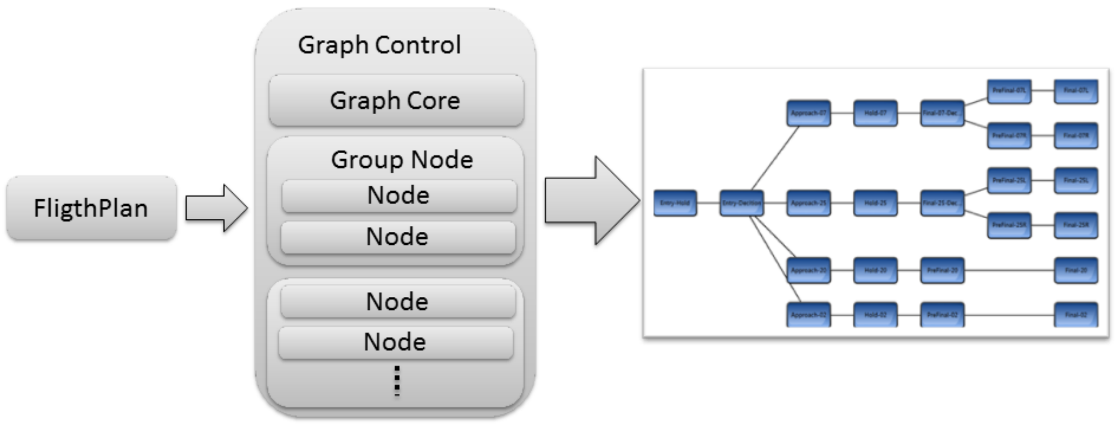


Figure 3.4: Architecture of control to display flight plan as a graph

Type	Name	Description
Method	PrintGraph	Paint the flight plan as a graph

Table 3.4: Input method of graph control

3.5. Video and Audio Player and Grabber

One of the requirements to be fulfilled by the ground station is to display in real time the video from the UAS on board camera. The .Net Framework does not provide adequate support for managing a video capture target. In order to obtain the video from video capture target it is required to use an external library. To solve this problem we used the DirectX.Capture library [12]. This library provides a friendly interface to use some of the options provided by the DirectShow library.[1]

3.5.1. Design and Implementation

In order to manage audio and video we have implemented a control called UcVideoControl. The control has as input parameters the audio and video configurations. With this data the UcvideoControl configures the DirectX.Capture library. We have decided to use the DirectX library because DirectShow provides a lot of options and it is very difficult to configure it.

Although the facilities that this library provides, we have determined to implement some interesting features. First we have added the possibility to record or not the displayed video. The library always record the video but if we use it for a long time the video becomes extremely heavy.

[1] DirectShow is a multimedia framework and API produced by Microsoft for software developers to perform various operations with media files or streams.

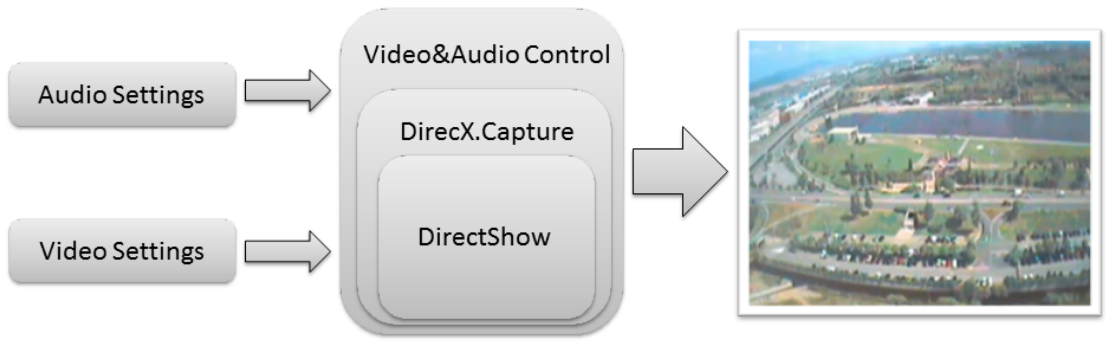


Figure 3.5: Architecture of video and audio control

The library provides a control to display the video designed in Windows Form. We have implemented all previous controls using WPF so that this control was not compatible. Now the UcVideoControl provides an interface to integrate Directx.Capture in WPF applications.

In order to manage this control we have implemented following methods (see Table 3.5):

Type	Name	Description
Method	Start	Start the video preview
Method	Stop	Stop the video preview
Method	EnableSaveVideo	Enable or disable the video recorder.
Field	AudioSettings	Set audio configurations.
Field	VideoSettings	Set video configurations.

Table 3.5: Methods to manage video and audio player

3.6. Interface for Joystick Management

The way of operating a UAS depends on its size and the configuration of the aircraft. In the case of conventional fixed wing aircraft, once an UAV has Take-Off, flight authority is mainly shifted to an autopilot on board. On the other hand, in the recovery phase, it is necessary to delegate the flight authority back to a remote pilot. The reason is that automatic approach and landing by only GPS based guidance may have large position errors. For safe landing within limited open space such as rooftops of buildings, school fields, and so on, more accurate guidance should be necessary. In this case, the flight control is switched back from autopilot(AP) to the pilot.

In order to manage the UAS in manual mode it is required to implement a control to load a joystick in GCS and to read its status. The .NET framework does not allow the developer to access directly to the joystick devices connected to the computer. So to access the joystick it is necessary to use an external library. DirectX.DirectInput [11] is a Microsoft Application Programming Interface (API) for collecting input from a computer user, via input devices such as the mouse, keyboard, joystick or other game controllers. It also provides a system for action mapping, which allows the user to assign specific actions to the buttons and axes of the input devices. This library is a powerful tool, but is very difficult to configure. With the purpose of facilitating the joystick management we have developed an interface called JoystickInterface.

3.6.1. Design and Implementation

This interface allows to manage the Joystick through the name of the device. The joystick interface returns the device connected in our PC. In order to update the joystick status we have implemented the Update method. When Update method is invoked, this interface sends at DirectX.DirectInput the order to update the device status. Then buttons and axis of joystick are read and stored in some fields (see Figure 3.6).

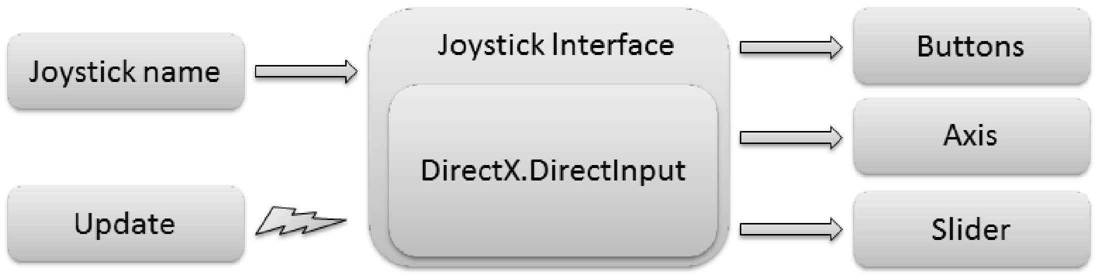


Figure 3.6: Joystick interface diagram

The methods and fields implemented in this interface are explained in Table 3.6:

Type	Name	Description
Method	FindJoysticks	Retrieves a list of joysticks attached to the computer. The list of joysticks as an array of strings.
Method	AcquireJoystick	Acquire the named joystick. You can find this joystick through the FindJoysticks method. Name of the joystick. The success of the connection.
Method	UpdateStatus	Update the properties of button and axis positions.
Method	ReleaseJoystick	Unaquire a joystick releasing it back to the system
Field	Buttons[]	provide the button state (true- pressed, false - released).
Field	Axis A-F	Provides the axis value on the joystick.
Field	Slider	Provides the slider value on the joystick.

Table 3.6: Methods to catch a joystick device and read its values

3.7. Google Earth Synthetic View

As seen before, we have developed a control to capture the video sent from the UAS. Sometimes the link of the video may not be available or we are simply running the work-stations in a simulated environment. To obtain some images that serve as reference control has been developed to provided a virtual image of what UAS is seeing. Using the UAS position, and Google Earth API [13] it is possible to generate a synthetic view.

The Google Earth API is a JavaScript library that allows developers to add Earth plug-in objects to their sites. This API allows the developer to configure most of the parameters of the Google Earth. It can be configured the camera point of view, the movements, the navigation controls, the layers, etc. The Google Earth API is oriented for web applications.

3.7.1. Design and Implementation

The Google Earth Synthetic View (called UcSVS) is the control developed with the purpose of generating a synthetic view for desktop applications. As you can see in Figure 3.7 the control has a two input parameters. The first parameter is the UAS telemetry, and the second one is the data provided from Google servers through Internet. In order to use this API in desktop applications, our control contains a web browser which loads an HTML page. This HTML page contains the JavaScript code to manage and load the Google Earth plug-in. Therefore we have implemented the control as a bridge between the code in C# and JavaScript code.

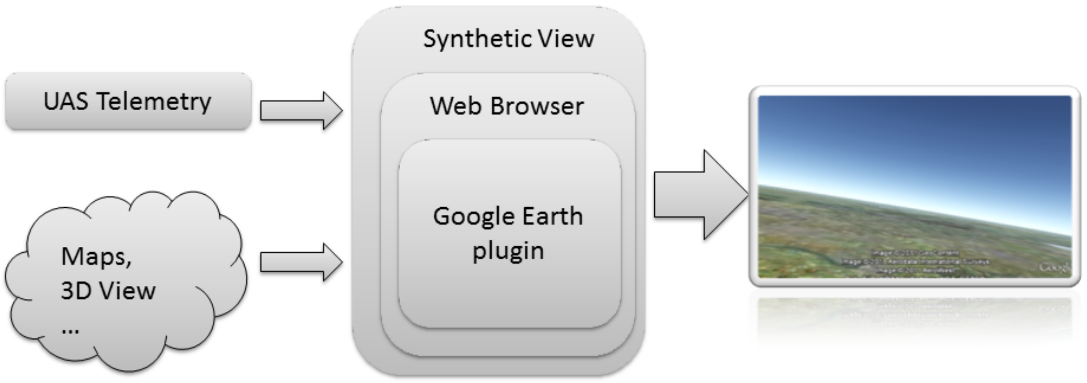


Figure 3.7: Architecture of control to generate synthetic view

The main problem of this control is that it require Internet connection, so it is especially designed to be used in simulation environments. To control the synthetic vision we have implemented some methods that allow us access to the Google Earth API methods (see Table 3.7).

Type	**Name**	**Description**
Method	StartControl	Load the control in web browser
Method	SetData	Update the camera status with UAS telemetry

Table 3.7: Methods to change the display of Google Earth plugin.

CHAPTER 4. FLIGHT PLAN MONITOR SERVICE

Due to a complexity of flight plan language developed in ICARUS research Group, it is necessary to implement an application to manage flight plans and UAS trajectory. This chapter presents the Flight Plan Monitor workstation (FPMo).

The Flight Plan Monitor (FPMo) is in charge of real-time monitoring of the entire UAS flight plan. In order to follow the UAS flight we need to locate the UAS and the flight plan on a map. This is the workstation complement to Flight Plan Manager (FPMa). The FPMo allows us to monitor and control the progress of the flight. It can send the flight plan to the FPMa, fly directly to a given leg, trigger emergency flight plan execution and update the flight plan. To monitor flight progress, it can display the flight plan structure with all flight plans (main and emergency), the FPMa status and the flight plan on the map.

4.1. Flight Plan Monitor Requirements

The minimum requirements of this software component are described below:

- The UAS and flight plan has been visualized over geo-referenced 2D-map.
- This application shows selected path or all possible path of loaded flight plan.
- It shows the flight plan in a graph view.
- The flight plan can be send to Air segment(FPMa).
- The workstation has to send the flight plan updates to Air segment.
- It has to send the Skip, GoTo and SetCondition command to Air segment(FPMa).
- It has to show the FPMa status.
- The FPMo has to provide a mechanism to edit the current flight plan and preview the modifications.

4.2. Flight Plan Monitor Architecture

Some controls as Geographic Information System and Display Manager and Graph Generator are used to make a Flight Plan Monitor Service (see Figure 4.1).

Geographic Information System and Display Manager is needed by the PoC to see the UAS trajectory and the flight plan over a map. Using the UAS telemetry and the UAS vector layer the path are displayed and the Poc can follow the UAS behavior. Also we have added the flight plan Layer to show and modify the flight plan. In order to do see the flight plan structure, the Graph control is added.

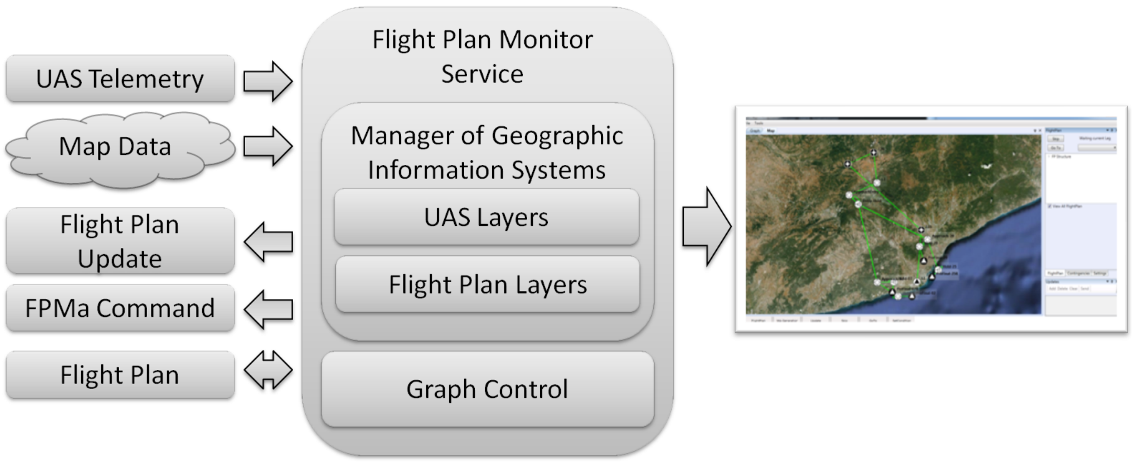


Figure 4.1: Architecture of Flight Plan Monitor Service

Due to the FPMo is a tool to exploit the Flight Plan Manager Service (FPMa) capabilities, both services have been communicated. The FPMo load the flight plan to the air segment and it allows execute some FPMa commands (Skip and GoTo commands and apply updates in flight plan).

4.3. Flight Plan Monitor Design and Implementation

In this section we describe the design and code implementation of the Flight Plan Monitor service. The Figure 4.2 shows the components of this application. Each component is the representations of each implemented class. For this reason the design and implementation are described in the same section.

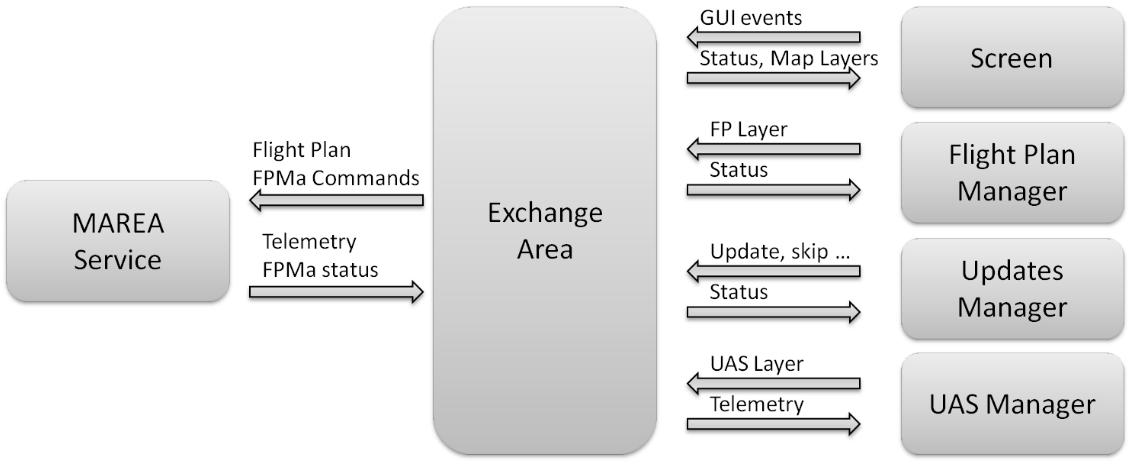


Figure 4.2: Design of Flight Plan Monitor Service

All components are described below:

- MAREA Service: MAREA service is the bridge between the network and the application. This component is implemented in FPMoService class. The MAREA service receive the data publish by the other services and it allows to send the commands to air segment.

- Exchange Area: The Exchange area is designed to share variables from different parts of the application. This class can be invoked from different parts of the layer, but sharing its value to every class. For example, there are values stored at objects that may be needed from different parts of the project. This component contains the UAS telemetry, the FPMa status and the status of each application component.

- Screen: The Screen is the main visual block and it is implemented in MapScreen class. This component contains the map control (see Section 2.3.1.), the graph control (see Section 3.4.) and the space for adding more controls. The components status and the result of the air segment commands are displayed in a MapScreen tool-bar . In order to place the controls, the Screen component contains the AvalonDock[1] control [17].

- Flight Plan Manager: This component manages all the task related with flight plan. The Flight Plan Manager configures the map control and load the flight plan vector layer. This component is implemented in a UserControl called FlightPlanManagement and controls the visualization of the flight plan parameters. This component, links the visualized flight plan parameters with the flight plan representation over the map. For example in order to visualize a leg information you can select the leg in the map and vice versa.

- Updates Manager: The Updates Manager implemented in UpdateManagement class, provides the tools to change the flight plan and update the flight plan in air segment. The UpdateManagement controls the previsualization of the flight plan changes and updates the flight plan when is changed by other application.

- UAS Manager: This piece adds a UAS layer in map control. The UAS manager use the UAS telemetry in order to shows the UAS position and its trajectory.

4.4. Flight Plan Monitor Graphical User Interface

This section is dedicated to explain the Graphical User Interface (GIU) of the Flight Plan Monitor service. The FPMo is implemented in one window in order to be used with one monitor. As shows in Figure 4.3, the application is composed by 4 areas.

The center area is dedicated for visualizing the flight plan representations. In this area is allocated the map control and the graph control in two different tabs.

The upper area is dedicated to allocate a menu with the basic functionalities of the FPMo workstation. The menu allows to open a flight plan, send flight plan to Flight Plan Manager Service and start the waypoints generation in FPMa service.

[1]AvalonDock is a WPF controls library which can be used to create a docking layout system like that is present in VisualStudio. It supports fly-out panes, floating windows, multiple docking manager in same window, styles and themes and it can host WinForms controls.

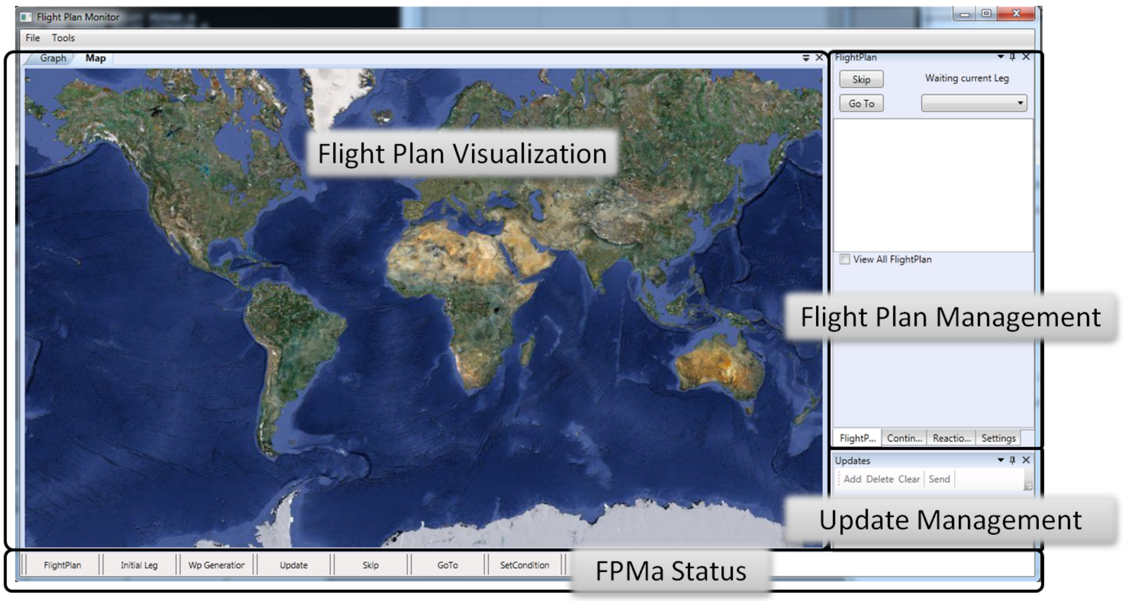


Figure 4.3: Flight Plan Monitor screen

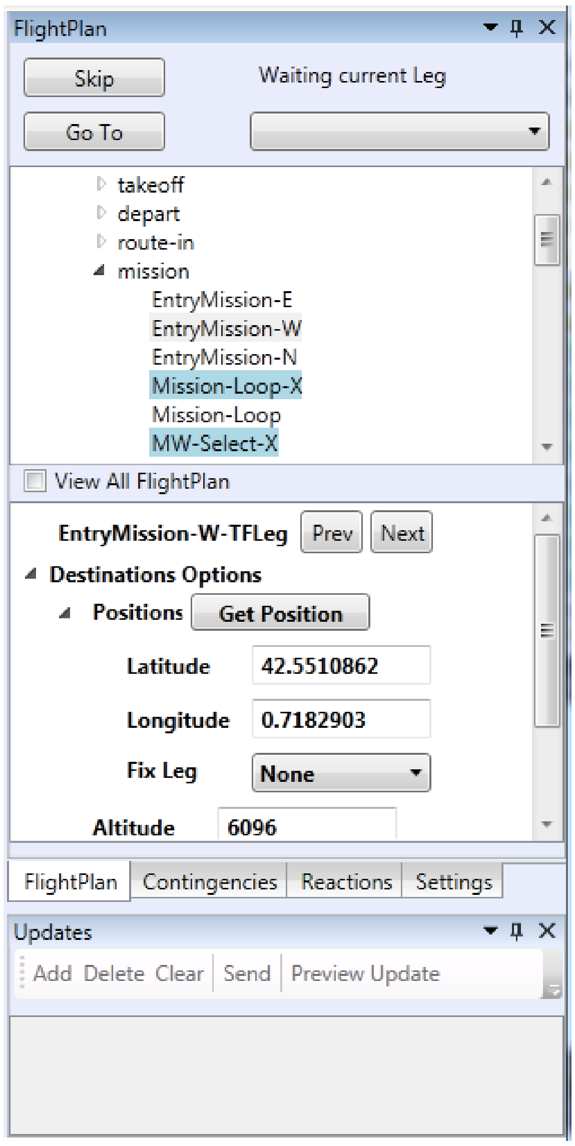
At the bottom of the application the user can see the FPMa status bar. It shows if any command sent to the FPMa have been completed. Each label has 4 states:

- Gray - Non command has been executed.
- Blue - The command has been sent but still the application has not received the ACK.
- Green - The command has been executed successfully.
- Red - The command has not been executed. An error has been produced in FPMa.

The tools to manage the flight plan are on the right side of the FPMo application (see Figure 4.4). This area is divided in three subareas. From top to bottom, in the first area we have the tool in order to send the Skip and GoTo commands to the VAS. With the Skip command, the UAS skip the current Leg and goes to the next Leg. Whit the GoTo command the PoC move the plane to another point in the flight plan.

The next area shows the flight plan composition and the information of each leg. The first control shows the flight plan as a tree, and when a flight plan item is selected, the information is showed in the second control. In this control the Leg information can be changed in order to update the flight plan.

Finally the area of the lower right corner allows to the user update the flight plan. In this area there are five buttons. This buttons allow add a leg in order to apply updates, send the last changes to FPMa and preview the updates in a map, or clear the changes applied after it can be send. FPManagement



Latitude 42.5510862

Longitude 0.7182903

Fix Leg None

Altitude 6096

Figure 4.4: Tools for flight plan and updates management

CHAPTER 5. FLIGHT MONITOR SERVICE

A pilot in command (PiC) is needed for UAS missions. Several comments in UAS guidance material and considerations such as [5] see the PiC as part of each UAS architecture. Therefore, an application is needed to solve all PiC needs. A real UAS flight has many different specific phases, and specific information needs to be shown for each of these phases. This application has to control the UAS mission states, telemetry, and flight plans, and receive some payload information.

Each workstation in the GCS has a different purpose. The Mission Monitor (MMo) is the workstation used to manage the mission and the payload. FPMa capabilities are managed with the Flight Plan Monitor (FPMo); by means of this workstation we can monitor and edit the flight plan. Finally, we need a workstation for the PiC in order to monitor the UAS flight in real time. This chapter focuses on the design and development of the workstation where the PiC is placed. This workstation is called the Flight Monitor (FM)

5.1. Flight Monitor Requirements

The main objective of this workstation is to provide an easy way to visualize the UAS information and provide to the Pilot in Command a Human Machine Interface (HMI) for controlling the UAS manually. During the design and implementation of the FM, certain requirements and specifications have been followed:

- The FM has to provide a flight information (altitude, speed, heading...).
- To provide a mechanism to control UAS manually
- The UAS and flight plan has been showed over a Map.
- The traffic information has been visualized.
- The video from on-board camera has been visualized.
- The workstation has to inform about the UAS alarms.
- The FM has to allow changing the VAS states.

5.2. Flight Monitor Architecture

In Flight Monitor(FM) workstation, we have used all the controls implemented in the ICARUS Control Library (see Figure 5.1). The FM service gets all information provided for the Virtual Autopilot Service in order to show the UAS behavior. The service allows update the VAS states for changing the UAS control.

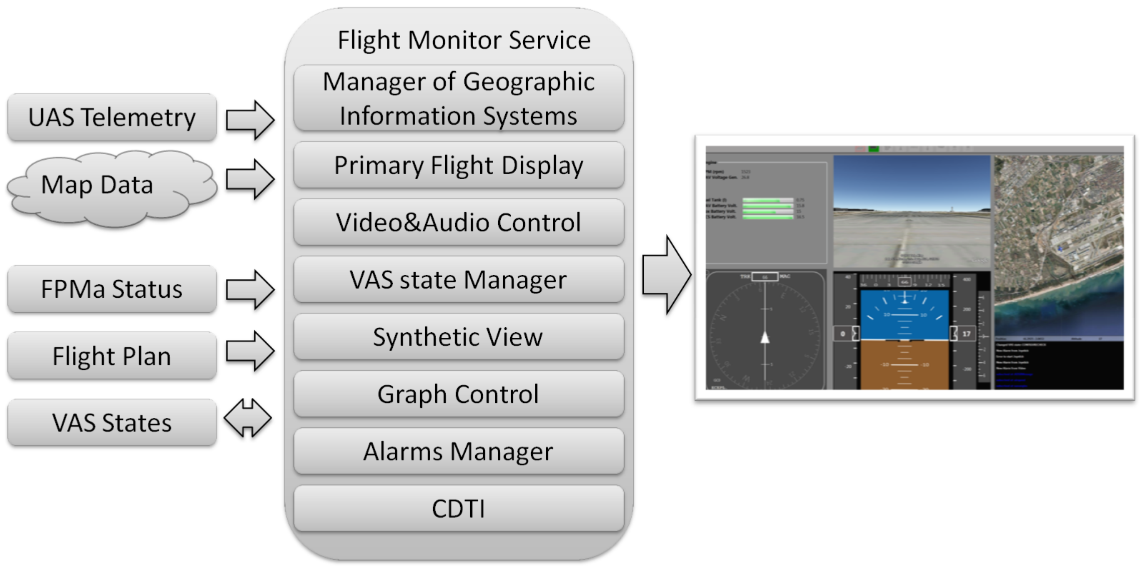


Figure 5.1: Architecture of Flight Monitor Service

Also the FM is communicated with the Flight Plan Manager(FPMa) and the Flight Plan Monitor(FPMo) for showing the flight plan over a map. The communication with the FPMa allows the flight plan synchronization. When the FPMo workstation change the flight plan, the FPMa informs if this change is valid. Then, the FM update the flight plan visualization.

Automatic Dependent Surveillance-Broadcast (ADS-B) Service, on the air segment, has the responsibility to send the traffic information over network. This information is received by the FM service and showed in CDTI control.

5.3. Flight Monitor Design

The Flight Monitor service as Flight Plan Monitor Service is centralized in a Exchange Area component (see Figure 5.2). The FM is composed by several block, which are described below:

- MAREA Service: MAREA service functions as a bridge between the network and the application.
- Exchange Area: As in the FPMo (see Section 4.3.), the Exchange area is designed to share variables from different parts of the application.
- Screen: The Screen is the main visual block. It works as a User Control container. The FM service can be composed more than one Screen. Each Screen have a layout for adding the differents controls.
- Control Manager: The Control Manager is the component responsible of managing the UserControls. The FM service is composed as Controls Manager as a data type

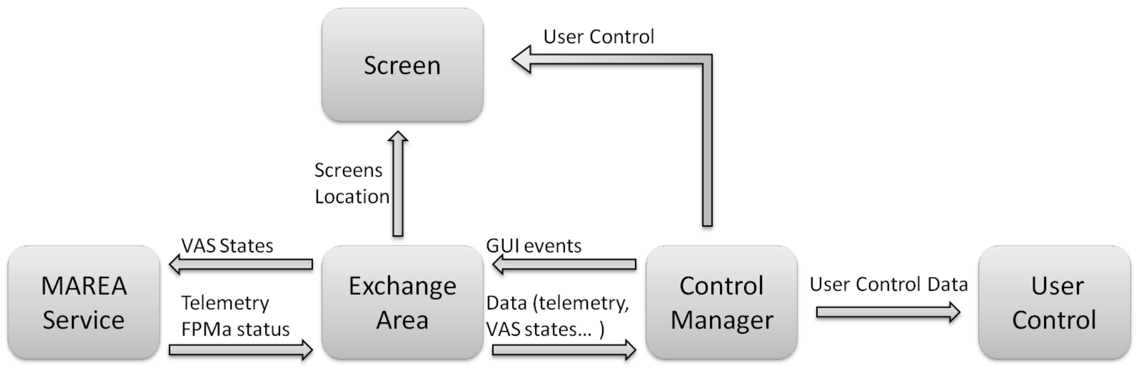


Figure 5.2: Design of Flight Monitor Service

shown in this service. For example there a Control Manager for telemetry, traffic information, video data, etc.

5.4. Flight Monitor Implementation

In this section we are going to describe the Flight Plan Monitor implementation. This section is divided in two subsections. One section is dedicated to describe the implemented classes and on the other subsection describes the graphical user interface.

5.4.1. Code Implementation

The Figure 5.3 shows the Flight Monitor main classes. In order to give an overview of code implementation we have been decided show and describe the most important classes of the FM.

- MainWindows: The MainWindows class is the main GUI class. The MainWindows is the container for adding all GUI components. This class contains the Graph control, the Map control and the alarms in order to notify the Flight Plan Manager Service status.
- SecondaryWindows: The SecondaryWindows class is a GUI class to set the VAS state control (see Section 3.3.) and another screen to configure the FM service. In the SecondaryWindows class the controls are embedded in a Tab control.
- TelemetryManagement: The TelemetryManagement class is in charge of managing the controls for visualizing the UAS telemetry. This class manage the UAVMapControlLayer (see Section 2.3.2.) the PFD (see Section 3.1.) control and the CDTI (see Section 3.2.) control. It opens a new thread in order to updates the PFD every 50 milliseconds and update the UAS position and CDTI control 2 times per second.

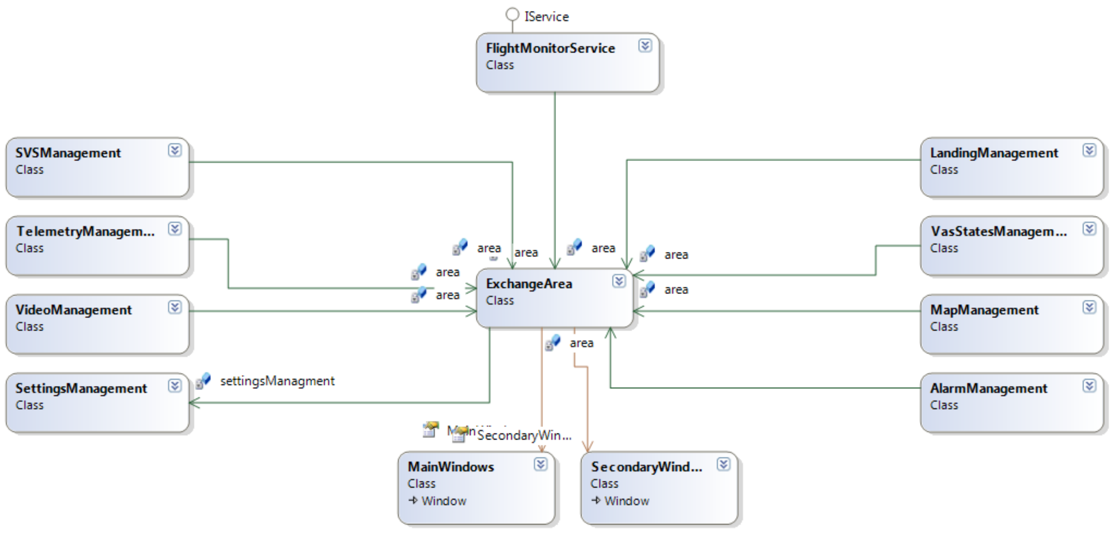


Figure 5.3: Class diagram of Flight Monitor Service

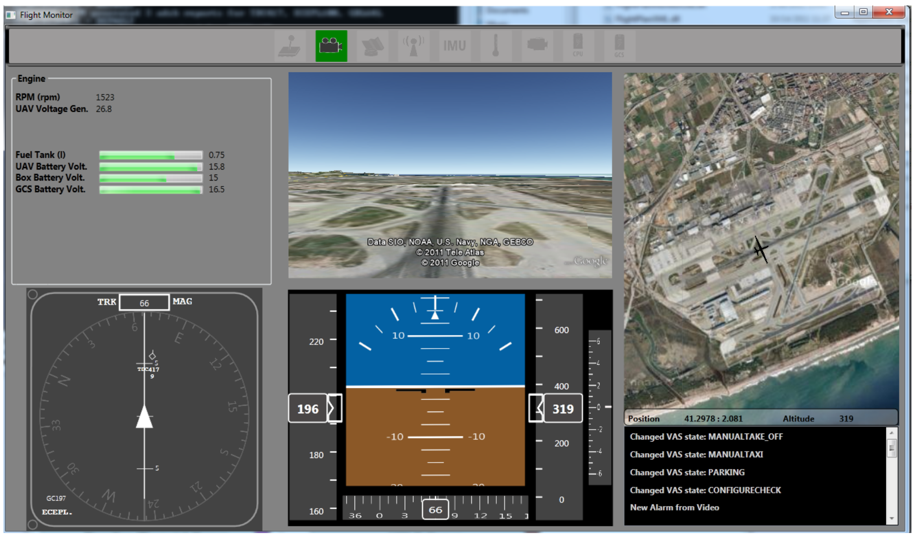
- VideoManagement: This class manages the Video Control (see Section 3.5.) and Synthetic view control (see Section 3.7.). It loads the appropriated control depending of the settings file. If the workstation does not have a video capture board connected it is possible to load a Synthetic view control to simulate on board video.
- MapManagement: The MapManagement class is in charge for visualizing the flight plan over a map. Also it is responsible to update the flight plan when it is changed for the Flight Plan Monitor Service.
- AlarmManagement: The AlarmManagement handless the components status. The alarms icons are changed depending of the components status. If the component works fine the color is green but if the control does not loaded properly the icon color is red. If the AlarmManagement has not received any notification, the icon color is gray.
- SettingsManagement: The SettingsManagement loads the settings file.

5.4.2. Interface Implementation

The Flight Monitor service is implemented using one or two windows in order to be used with one monitor or two monitors.

it can be executed in three modes, a Lite mode, One Screen mode and Two Screens mode. The Lite mode only loads the VAS State screens for changing the VAS states. It is the lighter version for deploying and debugging other services. In the One screen mode mode all controls are placed in main windows. The map control, video control, alarms and PFD are fixed and their can not be moved. In the left side of the application is located a tab host in order to place some tabs with engine controls, settings and VAS state control. Finally, the Two Screen mode is the default mode. The controls for showing the UAS information are placed in the main screen, and the controls to interact with the UAS are allocated in secondary window. The secondary window is designed to be used as touch screen.

The main window (see Figure 5.4) is designed to show the UAS information. It is composed by some controls of the ICARUS Controls Library. At right of the application, is located the map control. This map shows the UAS position and the flight plan. Below the control is placed a terminal to show execution information. In the center of the applications we can see the PFD control and the video control. The video control can be configured to show the on board video camera or it can be use the Synthetic View Control. The engine information and the CDTI are at left side of the main window.



Engine

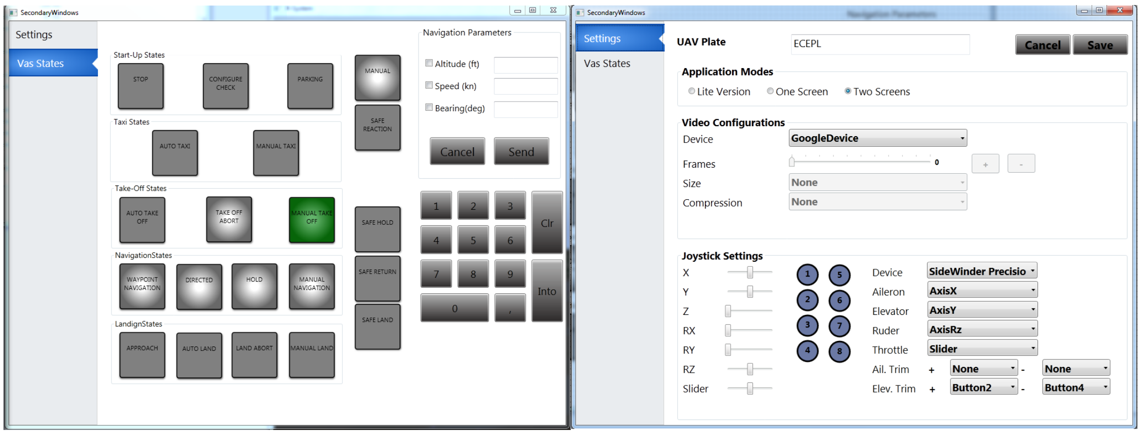
RPM (rpm)	1523
UAV Voltage Gen.	26.8
Fuel Tank (l)	0.75
UAV Battery Volt.	15.8
Box Battery Volt.	15
GCS Battery Volt.	16.5

Position 41.2978 : 2.081 Altitude 319

```
Changed VAS state: MANUALTAKE_OFF
Changed VAS state: MANUALTAXI
Changed VAS state: PARKING
Changed VAS state: CONFIGURECHECK
New Alarm from Video
```

Figure 5.4: Main screen of Flight Monitor Service

The secondary window (see Figure 5.5) is designed as an easy interface to manage the UAS. This window is composed by two screens. The VAS State screen and the Settings screen. The VAS State screen allows managing the UAS in Directed VAS state. In this state the PiC sends a course (altitude, speed and heading) to the UAS. The Settings screen it can be used in order to configure the starting mode of the application, the video source, and the joystick device.



UAV Plate: ECEPL

Application Modes: ○ Lite Version ○ One Screen [✓ Two Screens]

Video Configurations

Device	GoogleDevice
Frames	0
Size	None
Compression	None

Joystick Settings

Device	SideWinder Precisio		
Aileron	AxisX		
Elevator	AxisY		
Ruder	AxisRz		
Throttle	Slider		
Ail. Trim	+ None	-	None
Elev. Trim	+ Button2	-	Button4

Figure 5.5: Secondary screen of Flight Monitor Service

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

This chapter discusses the development of the project and the personal work. It has to take into account the initial objectives and requirements in order to compare them with the final result. After that we describe the future work that must be done in the workstations. Finally there is an environmental impact description for the project.

6.1. Conclusions

Once the application has finished, it is necessary to review the initial objectives. For doing the conclusion we are going to compare the final result of the applications with the initial objectives.

- The current Flight Plan Monitor and Flight Monitor do not have a serious errors. It is still early to ensure a correct operation in a real scenario but it is enough robust for a simulated environment.
- We have developed a library with the main controls of the applications. The workstations are developed using the controls implemented in ICARUS control library. Some controls are use in both applications, therefore the components have been reused.
- Using a library designed to display maps we have success in developing a control for visualizing georeferenced data over a map. This control creates a local cache for these maps and the maps are showed without Internet connection.
- The application has been tested under ISIS environment and the workstations and are properly integrated with other services.

After working in the ground control station for about two years, it makes me feel proud showing a complete, stable and functional version of two workstations. Especially I am proud to Flight Plan Monitor workstation.

When I started in the ICARUS group, my first work was developing a little program for showing maps without Internet connection. After two years and after I have been deleted and re-start the application many times, finally we have a release of the Flight Plan Monitor and Flight Monitor workstations. These workstation are going to be presented to Eurocontrol next November 17th as part of a big project.

Also I am so grateful of having the opportunity to learn about WPF. When I started in ICARUS research group, they give me the opportunity for learning how to develop user interface with WPF toolkit. Then, the WPF was a new tool for me and for the group. Two years later, I have learn a lot of things about C#. Now WPF is the default technology to develop new applications in ICARUS research group.

It has been a pleasure for me to have been involve in the ICARUS research group.

6.2. Future Work

Next are listed the tasks that can be implemented as a future work:

- Flight Monitor
  - Complete the take-off and landing views: The application is prepared to show all the necessary information of these flight phases.
  - Complete the alarms panel: The alarm panels are not finished yet. When the contingency service will be finished more alarms will appear.
  - Improve CDTI control: The current CDTI only shows traffic information. This control should also display information about contingencies but at moment there is not a service for sending contingency information.
  - Develop the engine and electrical information displays: In the current version of the platform there is not a service that produces this kind of information. So that, the current views are static, they cannot show any real information.
- Flight Plan Monitor
  - Support for Emergency flight plans: The specifications of the flight plan can incorporate emergency flight plans associated with the flight plan. Now ICARUS group is working to integrate the emergency plans in the simulator platform.
  - Support for multiples map layers: Now the user can load map data from a single server. It is interesting to load data from different servers at the same time such as weather, power lines, etc.
  - Fix some bugs of synchronism with vector layers: Sometimes, for a moment, the vector layers are not properly synchronized with the map..

The most important work remains for the future is to test again and again the workstations in order to be used in a real scenario.

6.3. Environmental Impact

As it can be seen, the project is the development of a software application. So that this has not directly environmental benefits, but the final goal of the ICARUS research has a completely environmental benefit.

The FM and FPMo will allow controlling a UAS mission over any kind of terrain and any time. If it is compared a mission operated by a UAS and a manned aircraft, there are many benefits for the UAS.

The use of a UAS benefits the environment by not using lots of fuel, it is much quieter, it does not require staff on board, etc. The project also takes into account the future applications. The first application which ICARUS is working is to controlling, monitoring and surveillance of forest fires. In the future the system could be extended to the rescue and maritime surveillance, weather control, forest monitoring, fauna monitoring, etc.

BIBLIOGRAPHY

[1] Royo Chic, P. “ICARUS Simulation Integrated Scenario”. *(Technical University of Catalonia. Castelldefels. 2010): 137–151.*

[2] Santamaria Barnadas, E. “Flight Plans Specifications Lenguage”. *(Technical University of Catalonia. Castelldefels. 2010): 33–48.*

[3] López Rubio, J. “Service Oriented Architecture for Embedded (Avionics) Applications”. ICROS-SICE International Joint Conference 2009

[4] López Lievana, B. Master thesis: *Ground Control Station for UAS over ICARUS System*. (Technical University of Catalonia. Castelldefels. 2009)

[5] “Guidance material and considerations for unmanned aircraft systems.”. RTCA. 2007 (March). Do-304.

[6] BruTile - GIS tiling library.
”**http://brutile.codeplex.com/**”

[7] SharpMap - Geospatial Application Framework for the CLR.
”**http://sharpmap.codeplex.com/**”

[8] Mapsui - Library for mapping.
”**http://mapsui.codeplex.com/**”

[9] Paparazzi UAV.
”**paparazzi.enac.fr/**”

[10] WPF Information.
”**http://www.wpftutorial.net/**”

[11] DirectX.Capture Class Library.
”**http://www.codeproject.com/KB/directx/directxcapture.aspx**”

[12] Microsoft.DirectX.DirectInput Class Library.
”**http://msdn.microsoft.com/en-us/library/windows/desktop/bb318766%28v=vs.85%29.aspx**”

[13] API de Google Earth.
”**http://code.google.com/intl/es-ES/apis/earth/**”

[14] OGG Standards.
”**http://www.opengeospatial.org/standards/wms**”

[15] Google Enterprise.
”**http://www.google.com/enterprise/earthmaps/portable.html/**”

[16] API de Google Maps.
”**http://www.google.com/enterprise/earthmaps/portable.html/**”

[17] Avalondock Control.
”**http://avalondock.codeplex.com/**”