Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Contingency Manager for ICARUS Simulated Integrated Scenario

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTORS: Oriol Caro Ignacio
Juanjo Rodríguez Carvajal

DIRECTORS: Pablo Royo Chic i Juan Manuel Lema Rosas

DATA: 9 de març de 2010

Título: Contingency Manager for ICARUS Simulated Integrated Scenario

Autores: Juanjo Rodríguez y Oriol Caro Ignacio

Directores: Pablo Royo Chic y Juan Manuel Lema Rosas

Fecha: 9 de marzo de 2010

Resumen

Asegurar la fiabilidad de un UAS en vuelo forma parte de uno de los temas más complicados que rodean al mundo de los aviones no tripulados. El motivo principal de esta preocupación es la cantidad de factores externos e internos del avión que entran en juego.

La pérdida de control del UAS causada, por ejemplo, por una lluvia intensa, un fuerte viento o, simplemente, un fallo del propio motor o del sistema eléctrico del avión, seria evitable si un gestor de contingencias estuviera disponible. Un gestor de contingencias se activa cuando un problema es detectado con el objetivo de garantizar la fiabilidad del sistema.

El objetivo principal de nuestro proyecto es realizar un gestor de contingencias para poder integrarlo en el ISIS (ICARUS Simulated Integrated Scenario). En esta plataforma existen proyectos dedicados al desarrollo del software de un UAS, pero ninguno de ellos desarrolla un gestor de contingencias.

Para poder implementar el gestor de contingencias, necesitamos desarrollar previamente simuladores que nos proporcionen los datos de los componentes que forman un UAS. Los simuladores que se van a implementar son los siguientes: uno de condiciones climáticas, otro del motor del avión y un último del sistema eléctrico del UAS.

Estas tres aplicaciones nos permitirán simular situaciones de emergencia; además de gestionar datos eléctricos, mecánicos y climáticos dentro de la plataforma de simulación del grupo ICARUS. Dependiendo de las alarmas que se recojan por el gestor de contingencias, de forma totalmente independiente, clasificará las contingencias y determinará la acción a realizar por el UAS. De este modo, garantizará en todo momento la seguridad del avión.

Title: Contingency Manager for ICARUS Simulated Integrated Scenario

Authors: Oriol Caro Ignacio and Juanjo Rodríguez Carvajal

Directors: Pablo Royo Chic and Juan Manuel Lema Rosas

Date: March, 9th 2010

Overview

Ensuring the reliability of a UAS in flight is part of one of the most complicated issues surrounding the world of unmanned aircraft. The main reason for this concern is the amount of external and internal factors of the aircraft involved.

The loss of control of the UAS caused, for example, due to heavy rain, strong wind or simply a failure of the engine itself or the airplane's electrical system, would be avoidable if a contingency manager was available. A contingency manager is activated when a problem is detected with the objective of ensuring system reliability.

The main goal of our project is to make a contingency manager to integrate it into the ISIS (Integrated Scenario Simulated ICARUS). There are projects on this platform dedicated to software development of a UAS, but none of them develops a contingency manager.

In order to implement the contingency manager, we need to develop previously simulators that provide us with information of the components that form a UAS. The simulators that are being implemented are: one of weather, other of the aircraft engine and the last is a simulator of the electric system of the UAS.

These three applications will allow us to simulate emergency situations in addition to managing data electrical, mechanical and climatic within the simulation platform of ICARUS group. Depending on the alarms collected by the Contingency Manager, completely independently, will classify the contingencies and will determine the action to be performed by the UAS. Thus, at all times will ensure the safety of the aircraft.

INDEX

LIST OF FIGURES

NOMENCLATURE

FG	FlightGear
OpenGL	Open Graphics Library
API	Application Programming Interface
ECMA	European Computer Manufacturers Association
ISO	International Organization for Standardization
POO u OOP	Object-oriented Programming
UDP	User Datagram Protocol
TCP	Transmission-Control-Protocol
XML	Extensible Markup Language
CHT	Cylinder Head Temperature
EGT	Exhaust Gas Temperature
RPM	Revolutions per minute)
OLP	Oil Pressure
OLT	Oil Temperature
FL	Fuel Flow
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
BOOTP	Bootstrap Protocol
DLL	Dynamic Link Library
UAV	Unmanned Aerial Vehicle
USAL	UAS Service Abstraction Layer
MAREA	Middleware Architecture for Remote Embedded Applications
UAS	Unmanned Aerial System
ICARUS	Intelligent Communications and Avionics for Robust Unmanned Aerial Systems
CM	Contingency Manager
HM	Health Monitor
CIC	Contingency Intelligent Control
CMP	Contingency Manager Panel
FTS	Flight Terminator System
ISIS	ICARUS Simulation Integrated Scenario
VV	Virtual Vehicle
VAS	Virtual Autopilot System
FPM	Flight Plan Monitor
ELM	Electrical Manager
EFMS	Engine and Fuel Manager System
WM	Weather Manager
MMA	Mission Manager
EM	Engine Manager
ELM	Electrical Manager
FPMS	Flight Plan Manager Simulated

SECTION 1. INTRODUCTION

Ensuring the reliability of a UAS in flight is part of one of the most complicated issues surrounding the world of unmanned aircraft. The main reason for this concern is the amount of external and internal factors of the aircraft involved.

The loss of control of the UAS caused, for example, due to heavy rain, strong wind or simply a failure of the engine itself or the airplane's electrical system, would be avoidable if a contingency manager was available. A contingency manager is activated when a problem is detected with the objective of ensuring system reliability.

The main goal of our project is to make a contingency manager to integrate it into the ISIS (ICARUS Simulation Integrated Scenario). There are projects on this platform dedicated to software development of a UAS, but none of them develops a contingency manager.

In order to implement the contingency manager, we need to develop previously simulators that provide us with information of the components that form a UAS. The simulators that are being implemented are: one of weather, other of the aircraft engine and the last is a simulator of the electric system of the UAS.

These three applications will allow us to simulate emergency situations in addition to managing data electrical, mechanical and climatic within the simulation platform of ICARUS group. Depending on the alarms collected by the Contingency Manager, completely independently, will classify the contingencies and will determine the action to be performed by the UAS. Thus, at all times will ensure the safety of the aircraft.

Therefore, we can define as primary objectives of our project:

- Design, implement and integrate into the ISIS a Contingency Manager that can collect the UAS alarms, classify these contingencies and determine the action to be performed by the UAS.

- Design, implement and integrate into the ISIS, different simulators that are able to receive data from the components of a UAS.

This project presents several personal motivations. While we were coursing several subjects of the career, we center our attention on the programming, the computing architecture, data transmission, and the hardware and software manipulation. We wanted to work on a group that accomplishes all these requirements. For this reason, we decided to work at the ICARUS research group.

We find very interesting to be involved inside a research group. We can learn how they work, see how it develops a large project which is composed of many parts implemented by different people.

We also consider the important fact to learn to write a formal document in English. Learn to do this kind of documents will be very useful in our future careers.

We consider that design and implement a contingency manager for unmanned aircraft is an interesting, and complete project to end our career.

We have considered that to achieve complete the project in an organized way and in a time not too long, it was necessary to perform a task planning. To achieve this, we made the following Gantt diagram that shows the project timing.

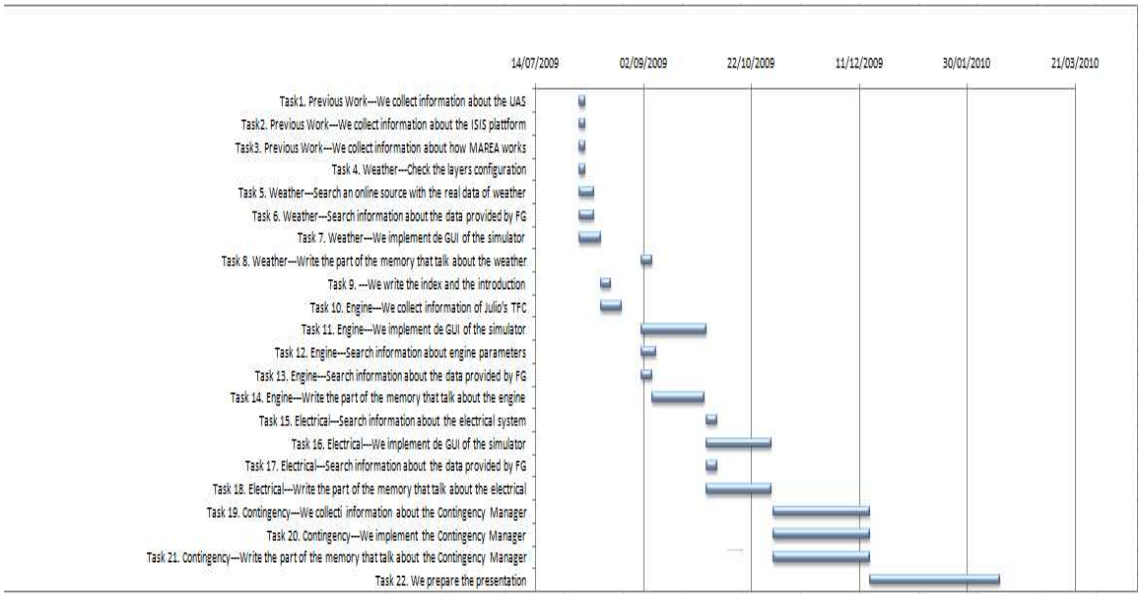

Fig.1.1 Project Timing.

During the first few weeks we will be devoted to collect information about all that surrounds our TFC. We collect information about the UAS, about the ISIS platform and about MAREA works.

Once these tasks will be finished, we are going to develop the first simulator.

To develop the Weather Simulator, first we collect information and later we design and implement the simulator.

This procedure is which we will follow to develop the Electrical Simulator, the Engine Simulator and the Contingency Manager.

Finally, to finish de TFC, we are going to prepare the presentation.

Then to finish this section we are going to talk about the document organization. The document is organized as follows.

Section II provides an overview about the previous work in the ISIS platform. It also explains what is a UAS and defines the middleware over we are going to work.

In Section III, IV and V present the three simulators: Weather Simulator, Engine Simulator and Electrical Simulator respectively. In these sections are explained which are the goals of these simulators, which requirements must reach and finally they also show the program results.

Section VI presents the Contingency Manager. In this section is described: which are the goals, requirements and architecture of the CM. Finally it also shows the program results.

Finally, Section VII concludes the document and proposes future work.

SECTION 2. PREVIOUS WORK

In this section we are going to describe how the ISIS platform is composed and how to we are going to integrate our project into it.

2.1 ICARUS Simulated Integrated Scenario (ISIS)

Definition and Overview

UAS are becoming one of the main assets to be employed in remote sensing applications both in civil and scientific environments. However, the current limitations when using non-segregated airspace makes extremely difficult to have available flight time to extensively test all subsystems required to achieve a specific mission.

Using real flights to test the complete UAS mission infrastructure involves high costs and risks. Current legislation requires an area of controlled and segregated airspace to perform the test flights. The UAS, its associated maintenance systems and the ground station need to be moved to the assigned airspace. Except for the smaller UAS, lots of personnel involved in the UAS mission also need to be mobilized. In addition, government permissions and adequate insurance coverage for the UAS operation has to be obtained. Finally, the possibility of damaging or completely losing the UAS should be considered, especially when testing new components or subsystems. Obviously, all of these restrictions and their associated logistic costs in both time and money make real UAS flight a bad option for experimentation.

Then, simulation is a pushing requirement previous to the real flight campaigns. Extensive research and experimentation is available in the area of aircraft and autopilot simulation, software in the loop and even hardware in the loop. However, little or no research is available in the area of “mission” simulation or in the area of multi-vehicle simulation. Modeling such scenarios is becoming mandatory because the operation of UAS will become correlated to different types of aerial vehicles and even ground vehicles.

For all these previous reasons, it is needed a simulation platform able to cope with a variety of civil UAS missions with little reconfiguration time and overhead. This platform has to be capable of, not only simulate the behavior of the UAS from the mission point of view, but it has to be able to include additional vehicles each one modeled with different levels of granularity. However, not only vehicles but their mutual interaction and the interaction with their surrounding environment have to be available to simulation.

To accomplish this objective they have implemented a distributed simulation architecture in which vehicles and environment is simulated trough specialized pieces of software. An underlying service-oriented communication middleware provides the underlying infrastructure to easily implement the inter-vehicle

coordination and even the coordination with third party applications providing additional simulated vehicles or simulated environment. They called this platform ICARUS Simulation Integrated Scenario (ISIS).

The idea of ISIS is to minimize both the test development and validation cost, as well as to provide an easy migration of the software from the tested platform to the real flight platform.

The general goals of the ISIS are:

- First, ISIS provides an environment in which the USAL components or services already designed can interact with others being designed.
- Second, it provides an easier and safer way to test the mission application.
- Third goal is related to the testing of system with complex scenarios.
- Fourth, ISIS offers multiple level of detail simulation.

Architecture and Design

The key idea here is to have a component capable to simulate the aircraft flight. In order to achieve this goal, they need a “Flight Simulator” with a simulated autopilot capable of interacting with the USAL. The “Flight Simulator” selected was the “FlightGear”.

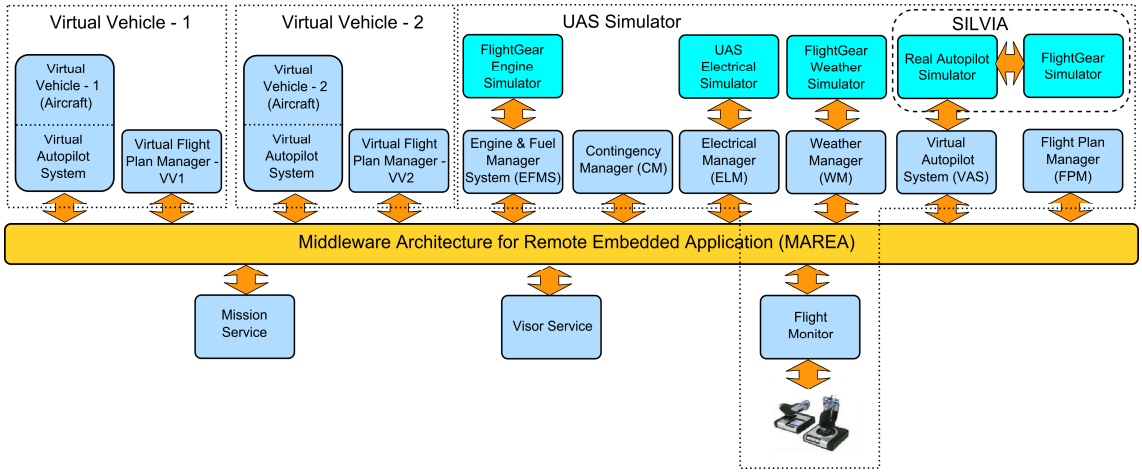

Fig.2.1 ISIS architecture.

In Figure 2.1 is showed one possible example of the ISIS architecture.

The UAS is composed by several services. The service composition depends on which sort of services or mission they want to test. In this example, we can see the Virtual Autopilot System (VAS), the Flight Plan Monitor (FPM) and the services that we are going to implement: the Contingency Manager (CM), the Electrical Manager (ELM), the Engine and Fuel Manager System (EFMS) and the Weather Manager (WM). The simulation versions of the ELM, EFMS and

WM are different from the full service version. The main difference is the source of the data. In this case, the data are provided by a simulator. In the full version, these data are supplied by the batteries sensors and the engine sensors respectively.

We will develop services CM, ELM, EFMS and WM. The latter three services will be simulators with the goal of creating emergency situations in the form of alarms. The CM will be responsible for categorizing these alarms to be processed correctly. There by achieving the stability of the UAV as far as possible. All these services belong to the MAREA middleware. For this reason can connect to other services if necessary.

One of the basic ideas of USAL architecture is the freedom to add or remove services from the final solution. Depending on which mission the UAS is going to deal, there are some service's blocks or others. Some are basic services, always needed. But others may differ depending on the mission objectives.

2.2 Middleware System Architecture (MAREA)

Middleware-based software systems consist of a network of cooperating components, in our case the services, which implement the business logic of the application and an integrating middleware layer that abstracts the execution environment and implements common functionalities and communication channels. In this view, the services are semantic units that behave as producers of data and as consumers of data coming from other services. The localization of the other services is not important because the middleware manages their discovery. The middleware also handles all the transfer chores: message addressing, data marshaling (so subscriber services can be on different platforms than the publisher service), delivery, flow control, retries, etc. Any service can be a publisher, subscriber, or both simultaneously. This publish-subscribe model virtually eliminates complex network programming for distributed applications.

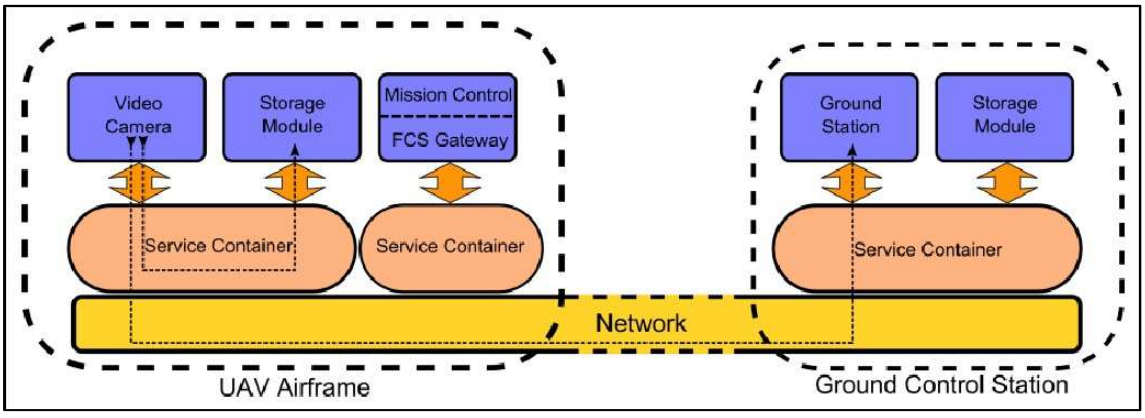

Fig.2.2 Middleware view of the UAV application

Each service of the middleware is able to publish variables and networking events, do a remote invocation and transfer files. The others MAREA's services can subscribe to these published items. Then we explain the differences between the two modes of publishing data, the remote invocation and files transmission.

As variables, we mean the transmission of structured, and generally short, information from a service to one or more additional services of the distributed system.

Events are similar to variables in the sense that both work following the publication-subscription paradigm. The main difference is that events, opposite to variables, 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.

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

In some cases, there is the need to transfer with safety continuous media. This continuous media includes generated photography images, configuration files or services program code to be uploaded to the service containers.

Implementation of our project in MAREA

For the implementation of our project we created four MAREA network services. The first three services correspond to each of the simulators that are planned. With them we have the possibility of publishing in the middleware all the simulated variables and the corresponding alarm events. When talking about alarms we refer to a notification to other services when variables of our aircraft are out of safety range and may pose a risk the aircraft airworthiness.

The last service implemented corresponds for the Contingency Manager. This service will be subscribed to three services listed above in order to assess the situation and make a decision. Once reached this point, we will be able to publish all other network services such decision

2.3 Unmanned Aerial Vehicle

An Unmanned Aerial Vehicle is an autonomous aerial vehicle capable of flying without any human pilot through a system of autonomous driving. It is called so (UAV) by the military of the United States, as it was the name given to the latest generations of aircraft capable of flying without a pilot on board.

The U.S is the country that has more applications and more of UAVs operating today. Is presumably as the power of board systems is increasing, the functions that will make the robots also grow. The use of UAVs today focuses on reconnaissance and surveillance missions.

The earliest example was developed after the First World War, and it was used during the Second World War to train operators of aircraft guns. However, until the late twentieth century was when operating the 'radio-controlled *UAV* through all the features of autonomy.

The unmanned aircraft may do works as important as the detection and monitoring of forest fires, disasters, whether natural or otherwise, and so on. It can also be applied in environments with high chemical toxicity and radiological type disaster Chernobyl, where it is necessary to sample at high risk of life and carry out environmental control. Aircraft comply with regulatory standards set forth in the Open Skies Treaty of 1992 that allow UAVs flying over the airspace of its signatories. They can cooperate in mission control drug trafficking and against terrorism. They could also record videos high quality to be used as evidence in an international trial.

The EPSC owns two UAV Shadow MK.1. This type of UAV can be seen in Figure 2.3. The Shadow MK.1 structure is made by Integrated Dynamics, and is a medium sized UAV. It is based on a classical twin-boom with engine in “pusher” position. Its stock engine is a 250cc boxer featuring about 22CV, which allows a maximum take-off weight (MTOW) of about 90kg.

The most likely application for MK.1 will be real-time detection, control and analysis of forest fires. Another possible application for it will probably be the calibration and supervision of VOR stations[1].

Fig.2.3 Shadow MK. 1

[1] VOR station: short of VHF Omni-directional Radio Range is one of the most used radio navigation aid for aircrafts and provides directional and range signal.

SECTION 3. WEATHER SIMULATOR

In this section we are going to explain all the features of the Weather Simulator and all the steps followed for its preparation.

This section starts with the Weather Simulator introduction. In this subsection, explains the definition of the application and its objectives.

This section follows with the Weather Simulator specification that it presents the functional goals of the service. Also, this subsection provides an overview of what we need to get a good simulator.

Next, the definition of the architecture and design of the Weather Simulator are discussed. It offers an explanation of the programming code and an outline of the structure of this program. It also presents an outline of the criteria for the design of the simulator.

The implementation of our project and the program results are depicted in the last subsection. This subsection details the implementation of the architecture and the design of the Weather Simulator.

3.1 Introduction

The Weather Simulator is a service capable of simulating weather conditions in the FlightGear (FG) and publishes all of them to the rest of the services. It is a service that lets you interact with the flight simulator through the exchange of data. Therefore, the user can modify the weather parameters that are simulated in the FG.

Weather conditions can be decisive in a real flight plan. A strong wind or an intense rain can cause serious problems to a UAS. The behavior of an UAS may vary significantly if weather conditions are not favorable.

In the ICARUS research group there is no service that considers the weather conditions. For this reason and to avoid all the problems explained in the previous paragraph, we need to create a Weather Simulator.

The goals of the Weather Simulator are the following:

- Simulate weather conditions at the flight simulator for that the user to perform a study of the UAS.

- Publish all weather conditions simulated to the rest of services of the ISIS platform. If some service of the platform needs some weather parameter may subscribe without problem.

- Generate and publish alarms to the rest of services when dangerous situations are simulated for the UAS. Some services will need these alarms to perform their tasks.

- Integrate the Weather Simulator in the ISIS platform. With this service, we increase the possibilities offered by the ISIS.

3.2 Specification

The Weather Simulator must be a service that allows the user:

- Alter all **weather conditions** simulated in the FG. These weather parameters are: wind, rain, snow, clouds, turbulences, visibility, temperature, dew point[2] and pressure. Among them we highlight those that directly affect the flight of a UAS. The highlights include: wind, rain / snow, turbulences and temperature.

- Determine the **elevation** at which we simulate the weather conditions. All parameters must be simulated within the elevation established. Elevation is important because not all UAS can fly to the same heights.

- Simulate weather conditions on different **atmospheric layers**. The user can simulate many weather conditions on the same stage if he is able to split the sky in layers. Thus, can be get a much closer simulation of reality.

- Choose the **reference level** for our simulation. The elevation, explained in the second point, can have the sea or the ground as reference level. Modify this information is important because the FG allows to simulate different atmospheric layers for each of them. If the user simulates with the sea as a reference level, the FG allows to the user to simulate five atmospheric layers. If the user simulates with the ground as a reference level, the FG allows to the user to simulate three atmospheric layers.

- Create an **easy** simulator and **friendly** to the user. Thus, the user can simulate weather conditions without being an expert in the field.

Finally, there is a graphic representation of the parameters simulated by the Weather Simulator. This representation can observe in Figure 3.1.

[2] Dew point: is the temperature to which a given parcel of air must be cooled, at constant barometric pressure, for water vapor to condense into water. The condensed water is called dew. The dew point is a saturation point.

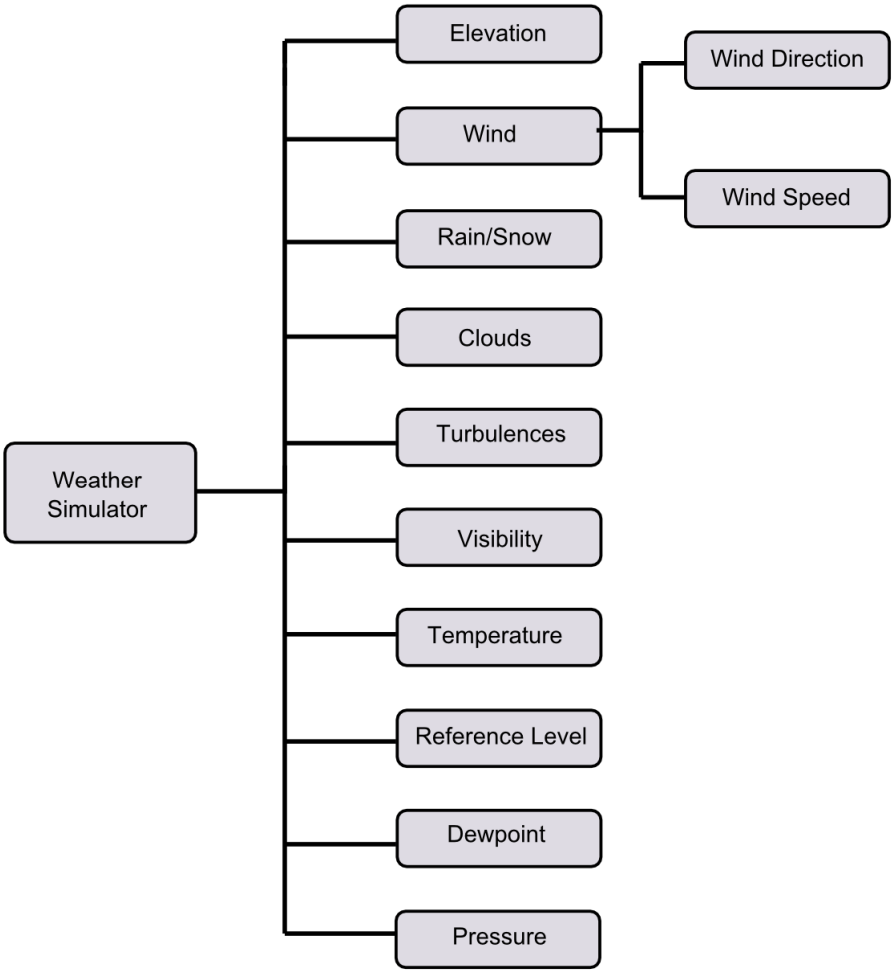

Fig.3.1 Parameters of Weather Simulator.

3.3 Architecture and Design

In the first part of the subsection will be referred to the program structure. It presents an overview of the architecture of the simulator.

The second part will be referred to the design of the Weather Simulator and all classes used for programming the service.

Architecture

Figure 3.2 shows the architecture of the programmed code of the Weather System. It can be seen that the simulator is divided into two layers.

The first layer, called FG Weather Simulator, consists of all classes relating to the graphical interface of the simulator. This group is composed by the

configuration forms. Also, it can be seen all those classes that allow communication with the FG.

The second layer, called Weather Manager (WM), consists of classes that allow publish data to other services. All classes of the WM will depend on a completely separate project called WeatherInterface. The interface is created to have a common definition of the classes. It is one way of sharing the definition of an object.

Specific definition of all classes that make up our project can be finding in the design subsection.

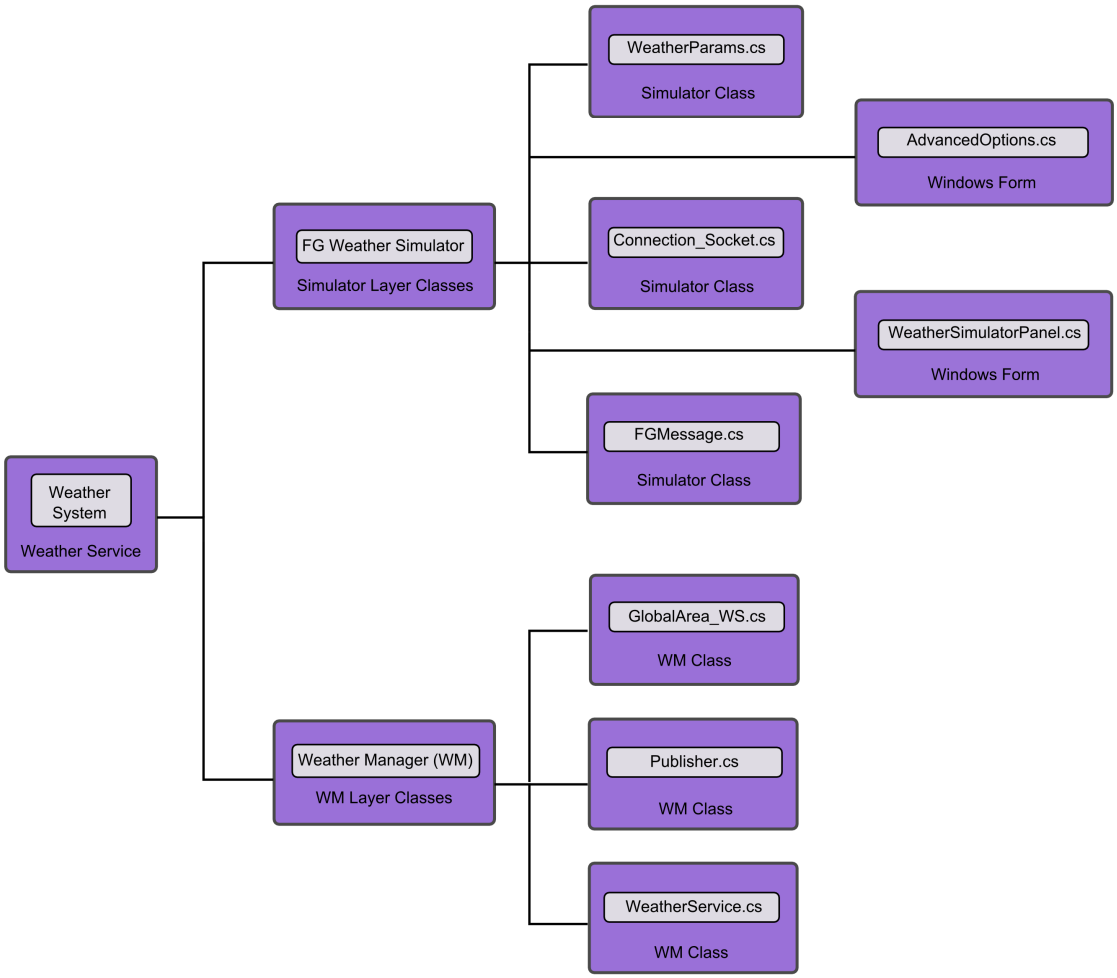

Fig.3.2 Weather Simulator Architecture.

The Weather System will modify when the hardware of the UAS of the ICARUS is finalized. From that moment, all weather conditions will detected by sensors. The UAS in flight will provide the necessary weather information. Thus, the Weather Simulator no will longer necessary. For this reason, we divided the Weather System in two layers. The FG Weather Simulator layer will be replaced by the layer of sensors. The WM is the only layer that could reused in the future.

Design

We have divided the section into two groups for explanation of the design of the simulator. The first, called *code design*, is responsible for explaining in detail each of the classes that have been used for programming the simulator code. The second, called *graphic interfaces design*, presents a general overview of the two designs that it has been used to create the graphical portion of the Weather Simulator.

Code Design

Next, we are going to explain in more detail each of the classes of the Weather System.

The FG Weather Simulator has the following classes:

- **Connection_Socket.cs**: This class establishes the socket connection between the Weather Simulator and the FG. It consists of two main methods. The first, called *send*, sends any type of string for a given port. This method is responsible for transmitting data from the Weather Simulator to the FG. The second method, called *reception*, receives the data that the FG sends through a configurable port.

  The sockets that connected to FG can send data using two protocols, UDP and TCP.

  The UDP protocol allows our simulator to run without the need of having established a previous connection to the FG. That is why this protocol has been chosen for all the sockets in our applications. The disadvantage of the UDP protocol is that it does not guarantee the arrival of all packets of the transmission. Information can be lost in shipping.

- **FGMessage.cs:** This class create the message to send to FG with the correct estructure. The FG needs that the messages, that it receives, follow a certain structure. Otherwise the flight simulator can not process the information transmitted. The message that the Weather Simulator sends with this class follows the following structure:

  41.288377 **\t** 53.78788878 **\t** 2.78878888**\n**

  As shown in the message example above, the values separated by \t represent values of different parameters of the flight simulator. The FG indicates that the message is finished with character \n.

  XML files are needed for that the connection between the Weather Simulator and the FG through messages be successful. The XML files relate the values of the message with the parameters of the flight simulator. The XML files can be input or output of information to the FG.

For the Weather Simulator we need two inputs XML files to enable us send to the flight simulator the information we want to modify. The first is called Weather_Aloft.xml and is responsible for relating the parameters with the FG when the sea is the reference level. The second is called Weather_Boundary.xml and, in this case, relates the parameters when the ground is the reference level. Both documents follow a structure like the one shown at Figure 3.3.

```
<?xml version = "1.0"?
  <PropertyList>
   <generic>
     <input>
        <! - Set autopilot control properties ->
          <line_separator> newline </ line_separator>
          <var_separator> tab </ var_separator>
            <chunk>
              <name> Elevation (ft) </ name>
              <type> float </ type>
              <node> / environment / config / aloft / entry / elevation-ft </ node>
            </ chunk>
```

XML code for connection between FG and Weather Simulator.

Fig.3.3 XML code of Weather Simulator.

As we can see in this XML code relates the *Elevation* variable of weather with the path simulator for the FG server. The flight simulator has a server where the user can observe the paths of all its parameters. Thus, with the XML file the user can relate the new values with the parameters of the FG. In the case of the Figure 3.3, we assign the variable *elevation* to the variable *elevation-ft* of the path */environment/config/aloft/entry.* We also determined *<type>* labeled the type of variable we are sending. This process is continuous for all variables submitted. The Weather Simulator sends two messages that are totally different. For this reason, we need two XML, each for a different reference level. The message we send is directly related to the XML used in transmission. For that we need to create two separate sockets, one for each of the messages.

- **WeatherParams.cs:** This class prepare the message of the FGMessage class with a given data. WeatherParams.cs determines with that values are going to create the message of the FGMesaage class. The values will be determined by WeatherSimulatorPanel class or AdvancedOptions class. This new class needs to reference constantly the FGMessage.cs class.

  The WetaherParams.cs consists of two methods. Each creates a different message. The first message is created when the user simulates weather situations with the ground as reference level. The second message is created when the user simulates weather conditions with the sea as reference level.

- **WeatherSimulatorPanel.cs:** This class is the main form throughout the application. It is the main graphic interfaces of the simulator. With this

class the user can interact with the Weather Simulator. In the *implementation* subsection is a more detailed explanation of this class.

- **AdvancedOptions.cs:** This class introduces the second form of the Weather Simulator. It is a graphic interfaces that allows users to create weather conditions much more concrete or specific. In the *implementation* subsection is a more detailed explanation of this class.

The Weather Manager (WM) has the following classes:

- **WeatherService.cs:** This class determines all the variables and events published by the weather system. If it is necessary, the user would have the posibility to subscribe to other services with this class. The WetaherService.cs also is responsible for initialize the WeatherSimulatorPanel.cs.

- **GlobalArea_WS.cs:** This class is like a box where there are all the values of variables and events to be publish. When the user simulates weather conditions, all parameters are stored in this new class. Thus, it will be take the values of the GlobalArea_WS.cs when it is necessary to publish data to the other services.

- **Publisher.cs:** This class is responsible for collecting the values of GlobalArea_WS.cs for publication in middleware. Thus, all services can use the parameters simulated by the weather simulator.

Graphic Interfaces Design

We used two designs for creating the Weather Simulator. The first design developed for this simulator was based on the way to represent the values of the FG weather panel (see Figure 3.4). The flight simulator shows values of wind, turbulence, visibility, among layers defined at different altitudes.

Fig. 3.4 Layer design FG

FG offers the possibility of splitting the sky in five layers from sea level and three from ground level. In each of these divisions we can determine a height and a series of meteorological parameters to choose. It allows to reflect the situation that we want to simulate accurately.

For proper operation of our first design, we must specify values for wind speed and direction, altitude, turbulence, visibility, temperature and pressure for each of the layers. To ensure a good simulation, the user should know exactly all the characteristics of the parameters that represent the simulator. Thus, we realized that it was not a fast and easy to use. It was a program where too many parameters to be modified to achieve a desired simulation.

Once this problem was noticed, we decided to rethink the simulator again and deploy it in a much more intuitive way. We would do much more visual simulator capable of recreating a real situation with a few mouse clicks, a precise direct application at a time. Thus we decided to implement a program where the main elements were the buttons.

The parameters of the wind direction and wind speed were reduced to a single group, *wind*. Thus, from three general conditions as long, medium or little we are able to represent the two characteristics of wind with a single button. These three conditions have also been applied to the parameters of *turbulence* and *visibility.*

The initial design has not been wasted but has been used as advanced options. Thus we can specify exact values for each of the parameters that are simulated.

3.4 Implementation

As a result we may find a simulator with two configuration panels: Weather Simulator Panel and Advanced Options. In this way we satisfy all user needs.

The Weather Simulator Panel is the main form of the Weather Simulator. With it, the user can simulate weather conditions quickly and accurately. The Advanced Options is the secondary form of the Weather Simulator. With it, the user can simulate much more specific weather conditions.

The following explains in greater detail the features and the functionalities of each of the forms.

3.4.1 Weather Simulator Panel

We represent the final implementation of Weather Simulator Panel in the Figure 3.5.

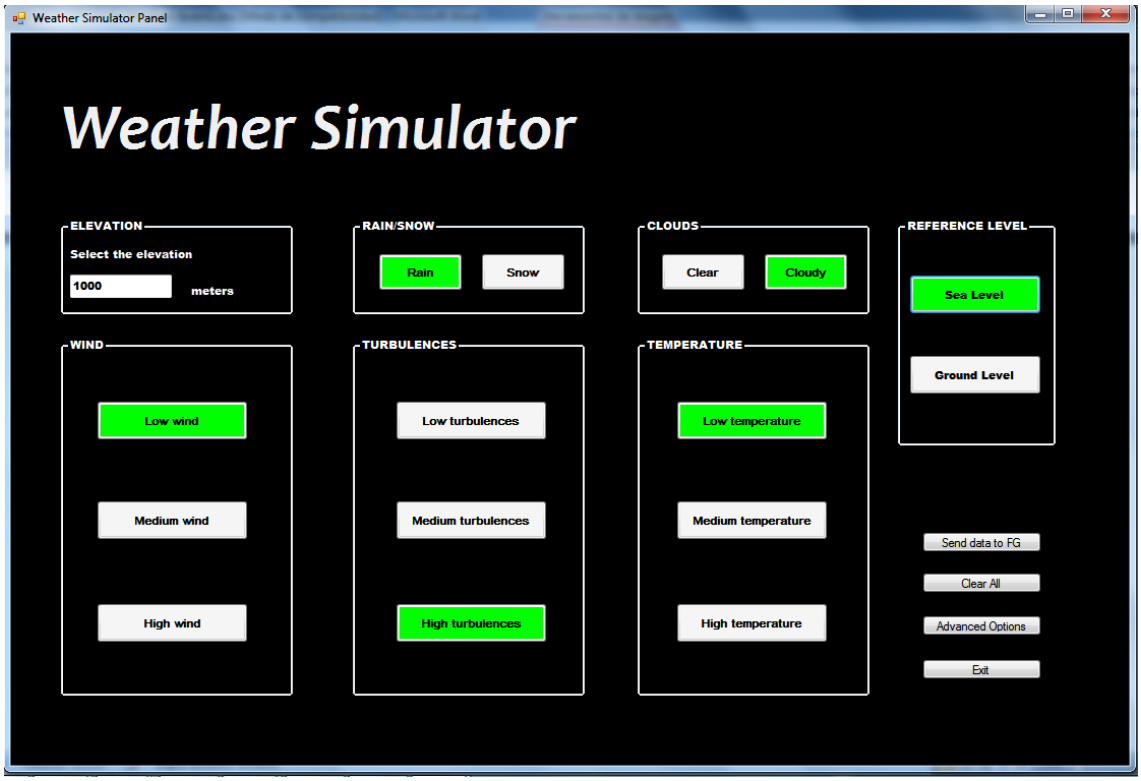

Fig.3.5 Weather Simulator Panel.

As it is shown in Figure 3.5, the Weather Simulator Panel is divided into different sections. Each section belongs to each parameters group that is simulated: wind, turbulence, temperature, reference level, altitude, clouds and rain or snow. The sections are clearly differentiated from each other. Every section is formed by their corresponding buttons.

Next, we are going to describe each weather sections from left to right of the Figure 3.5.

The first section that we are going to describe is the **elevation** section. By means this group, we specify the height at which we want to simulate a particular meteorological situation in one altitude layer. The limit or end values of this layer is determined by the values entered in the elevation section. All parameters of wind, turbulence, clouds and temperature that the user wants to represent are simulated in the range of altitudes of the layer.

The three sections that are explained below have three configuration buttons. These sections are those that are below the elevation section in Figure 3.5. With the three buttons that the sections have, the user can simulate three conditions for each section; short, medium and long. The values each section will depend on the condition to which they belong.

One of these three sections is the **wind** section. This value includes two characteristic parameters of the wind, the direction and the speed.

The wind direction parameter is important because the effects caused by the impact of wind in a certain direction are directly related to the orientation of the plane. The wind will affect the plane differently if the orientation of the plane and wind are the same or not. For the three conditions of this parameter a single value of direction is used. Sixty degrees has been used as a general value.

The velocity is a variable value. FG presents this parameter in kilometers per hour (km/h). Thus, we believe that 100 km/h was an appropriate maximum value for the simulation of the wind because the aircraft have important variations in their trajectories. Minimum value determined as 20 km/h, a value that does not affect the behavior of the aircraft-

Another section of the Weather Simulator Panel which involves setting three conditions is the section of **turbulence**. The *turbulence* section is to the right of the *wind* section in the Figure 3.5. The buttons of this section represent a value with range between 0 and 1. With the maximum value, we observe that the simulation results in a continuous drift of the aircraft even to lose all control over it.

The last section of the three sections is the **temperature**. The *temperature* section is to the right of the *turbulence* section in the Figure 3.5. This parameter allows the user to simulate a high temperature like a fire or a low temperature like the upper layers of the sky.

Next, can be seen the **rain/snow** section at top of the *turbulences* section and to the right of the *elevation* section in Figure 3.5. This section does not follow the pattern of previously explained. In this case, the section is formed by two buttons.

The section of *rain/snow* has been designed to simulate the raining or snowing. A heavy rain or snowfall affects the flight of the plane. It will rain with great intensity if the user selects the button *rain.* With the same intensity it will snow if the *snow* button is pressed.

Then, can be seen the **clouds** section to the right of the *rain/snow* section in the Figure 3.5. This section follows the pattern of the *rain/snow* section. In the section of clouds, we are able to simulate a clear or cloudy sky. In the Weather Simulator Panel are represented two types of clouds in the first layer of the atmosphere. This layer will be limited between the chosen reference level and the altitude selected in section *elevation*.

The last section to configure parameters is the **reference level** section. This section is to the right of the *clouds* section in the Figure 3.5. This value is very important in our simulator as it has the responsibility for determining the beginning of our atmosphere layer, where we represent all weather conditions chosen.

Finally, can be seen the **buttons** section under the *reference level* section in the Figure 3.5. The buttons of this section are: *send data to FG, clear all, advanced options* and *exit.*

With the first button, the *Send data to FG,* we initialize the data transmission. This data transmission is made through a socket UDP. For more information about the transmission see Appendix Section 1. With the *reset all* button, reset all the parameters previously set in the simulator. Thus, we can create new weather situations more quickly than restart the application. Using the *advanced options* button opens the panel that explained in the next section (see Figure 3.6).

3.4.2 Advanced Options

We represent the final implementation of Advanced Options of Weather Simulator in the Figure 3.6.

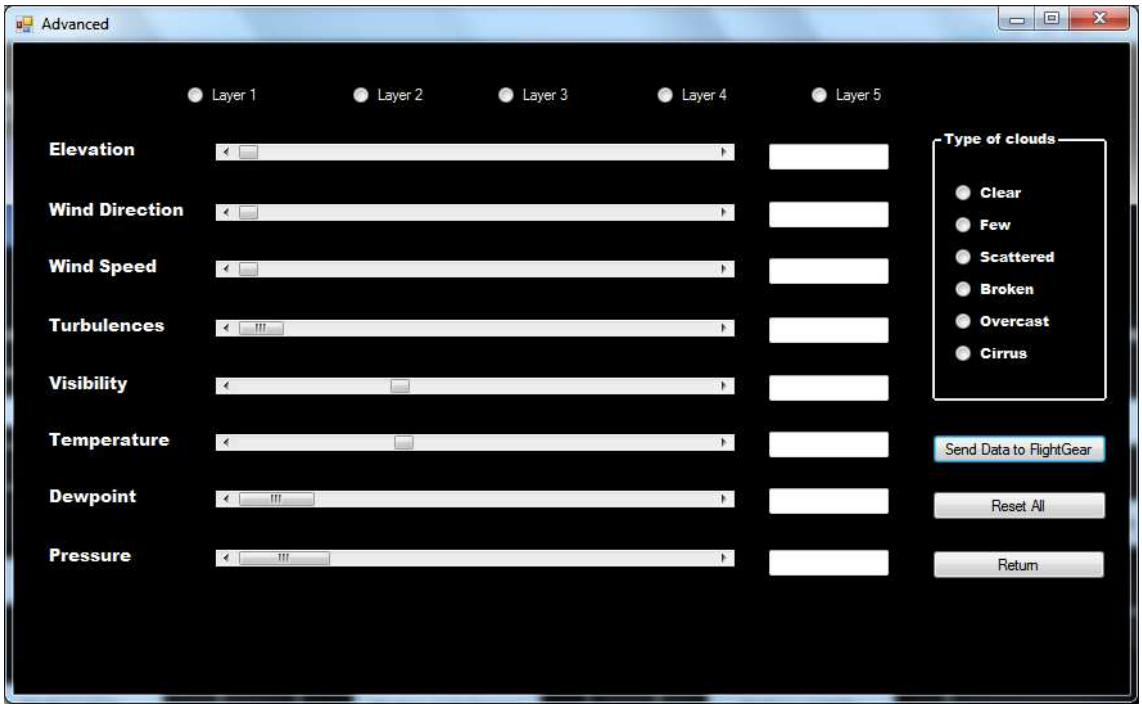

Fig.3.6 Advanced Options of Weather.

The Advanced Options is a form of the application created to achieve specific situations. With this new form, we are able to simulate climatic parameters much more concrete. This new panel allows the user to recreate a weather situation through layers of altitude. Now we are able to simulate all the parameters with the same configuration that offers the FG. Five atmospheric layers may be represented. The fact of adding this new feature will create a much more complete and detailed simulator. The new panel sends the information with the sea level as reference. This is because FG that it only provides the ability to simulate five layers with that reference level.

The Advanced Options consists mainly of four sections. All of which are interrelated to achieve the same goal, extend the simulator capabilities. Next, we are going to describe each *advanced options* sections from left to right of the Figure 3.6.

The first section that we are going to describe is the **scrolls** section. This section is comprised of eight scrolls and eight textbox. The eight scrolls are able to assign values to all parameters that are simulated. These parameters are: height, direction and wind speed, temperature, pressure, dew point and visibility. The eight textbox represent the values of the eight scrolls. These values are represented numerically and their corresponding unit.

Next, can be seen the **layers** section at top of the *scrolls* section in Figure 3.6. This section is comprised of five radiobuttons that allow us to select of which layer we want to send data. Thus, we can represent all the parameters listed above in all layers of existing elevation in FG.

Then, can be seen the **types of clouds** section to the right of the *scrolls* section in Figure 3.6. This section is comprised of six radiobuttons. Each of these radiobuttons represents a different type of cloud. The possibilities we have are the following: clear, few, scattered, broken, overcast, and cirrus. For each layer selected can simulate a different type of cloud.

The *few* option and the *scattered* option represent stratus type clouds. The *broken* option and the *overcast* option represent nimbostratus type clouds. The *cirrus* option represent cirrus type of clouds. For more information about clouds see Appendix Section 2.

The last section that can be seen in Figure 3.6 is the **buttons** section. This section is under the *types of clouds* section. The buttons that this section represents are: *Send Data to FG, reset all* and *return.* With the *send data to FG* button sends the information as the Weather Simulator Panel. With the *reset all* button reset the parameters configured. Finally, with the *return* button back to the main panel and saving the data set.

SECTION 4. ENGINE SIMULATOR

In this section we are going to explain all the features of the Engine Simulator and all the steps followed for its development.

This section starts with the Engine Simulator introduction. In this subsection, it is explained the definition of the application and its objectives.

This section follows with the Engine Simulator specification that it presents the functional goals of the service. Also, this section provides an overview of what we need to get a good simulator.

Next, the definition of the architecture and design of the Engine Simulator are discussed. It offers an explanation of the programming code and an outline of the structure of this program. It also presents an outline of the criteria for the design of the simulator.

The implementation of our project and the program results are depicted in the last subsection. This subsection details the implementation of the architecture and the design of the Engine Simulator.

4.1 Introduction

The engine simulator is an application that can emulate the engine system of a UAS. The user has the ability to visualize engine parameters of a UAS that are simulated in the FG. The Engine Simulator receives constantly the telemetry of the FG. Also, the user has the ability to recreate situations of failure of the parameters named above.

We need to create the Engine Simulator because the procedure of turning on the engine for each test of the simulation in a laboratory setting is an inappropriate option for experimentation. Also, the Engine Simulator is necessary because it provides the possibility of simulating engine alarms without compromising the real engine system.

The goals of the Engine Simulator are the following:

- Simulate the engine system in order to perform a study of the engine of the UAS.

- Publish all engine parameters simulated to the rest of services of the ISIS platform. If some service of the platform needs some engine parameter may subscribe without problem.

- Generate and publish alarms to the rest of services when dangerous situations are simulated for the UAS. Some services will need these alarms to perform their tasks.

- Integrate the Engine Simulator in the ISIS platform. With the Engine Simulator service, we increase the possibilities offered the services of the ISIS.

4.2 Specification

The Engine Simulator must be a service that allows the user:

- Emulate the most characteristic parameters of an engine system. These engine parameters are: RPM *(Revolutions per minute),* CHT *(Cylinder Head Temperature*), EGT *(Exhaust Gas Temperature),* oil pressure, oil temperature, fuel flow and fuel level.

- Visualize the parameters of the UAS engine of the FG. In this way, the user will know the actual data of the aircraft. These parameters will help the user to conduct the study of the UAS engine.

- Choose different values for the parameters that we are going to simulate. Thus, the application will simulate different types of alarms.

- Visualize what is happening. A graph for each of the simulated parameters can help the user to understand the simulation. Thus it is clearly differentiated the FG values and the new values of the Engine Simulator.

- Configure the application to any UAS. The user should be able to modify the values of the parameters that are simulated on the Engine Simulator. Thus, the simulator will be flexible for any type of aircraft.

- Create an **easy** simulator and **friendly** to the user. Thus, the user can simulate engine conditions without being an expert in the field.

Finally, there is a graphic representation of the parameters simulated by the Engine Simulator. This representation can observe in Figure 4.1.

These values refer to the engine of a Cessna 172. For more information about the Cessna see Appendix Section 3.

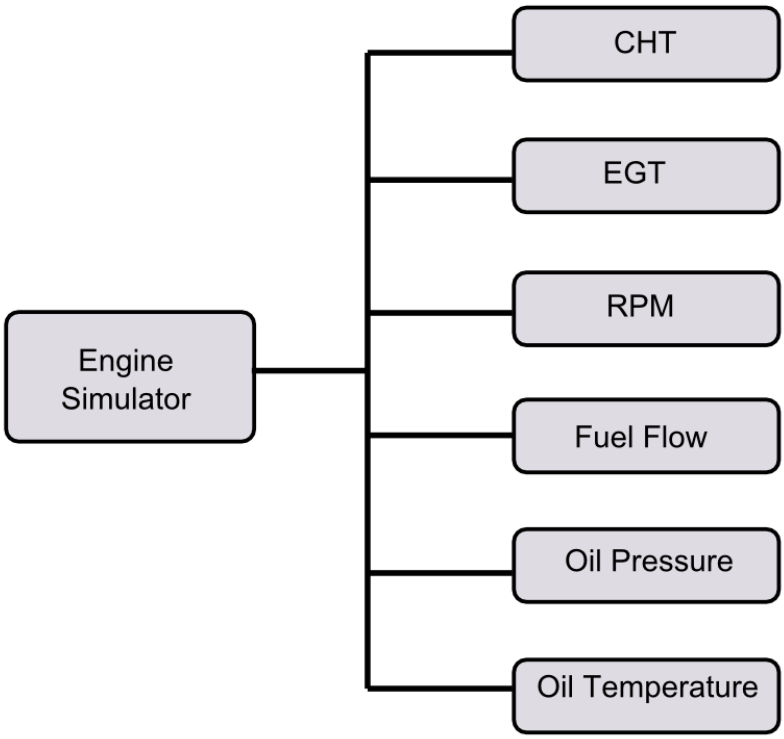

Fig.4.1 Parameters of Engine Simulator.

4.3 Architecture and Design

The first part of the subsection will be referred to the program structure. It presents an overview of the architecture of the simulator.

The second part will be referred to the design of the Engine Simulator and all classes used for programming the service.

Architecture

Figure 4.2 shows the architecture of the programmed code of the Weather System. It can be seen that the simulator is divided into two layers.

The first layer, called FG Engine Simulator, consists of all classes relating to the graphical interface of the simulator. This group is composed by the configuration forms of the simulator. It can also be seen all those classes that allow communication with the FG.

The second layer, called Engine Manager (EM), consists of classes that allow publish data to other services. All classes of the EM will depend on a completely separate project called EngineInterface. The interface is created to have a common definition of the classes. It is one way of sharing the definition of an object.

Specific definition of all classes that make up our project can be finding in the design subsection.

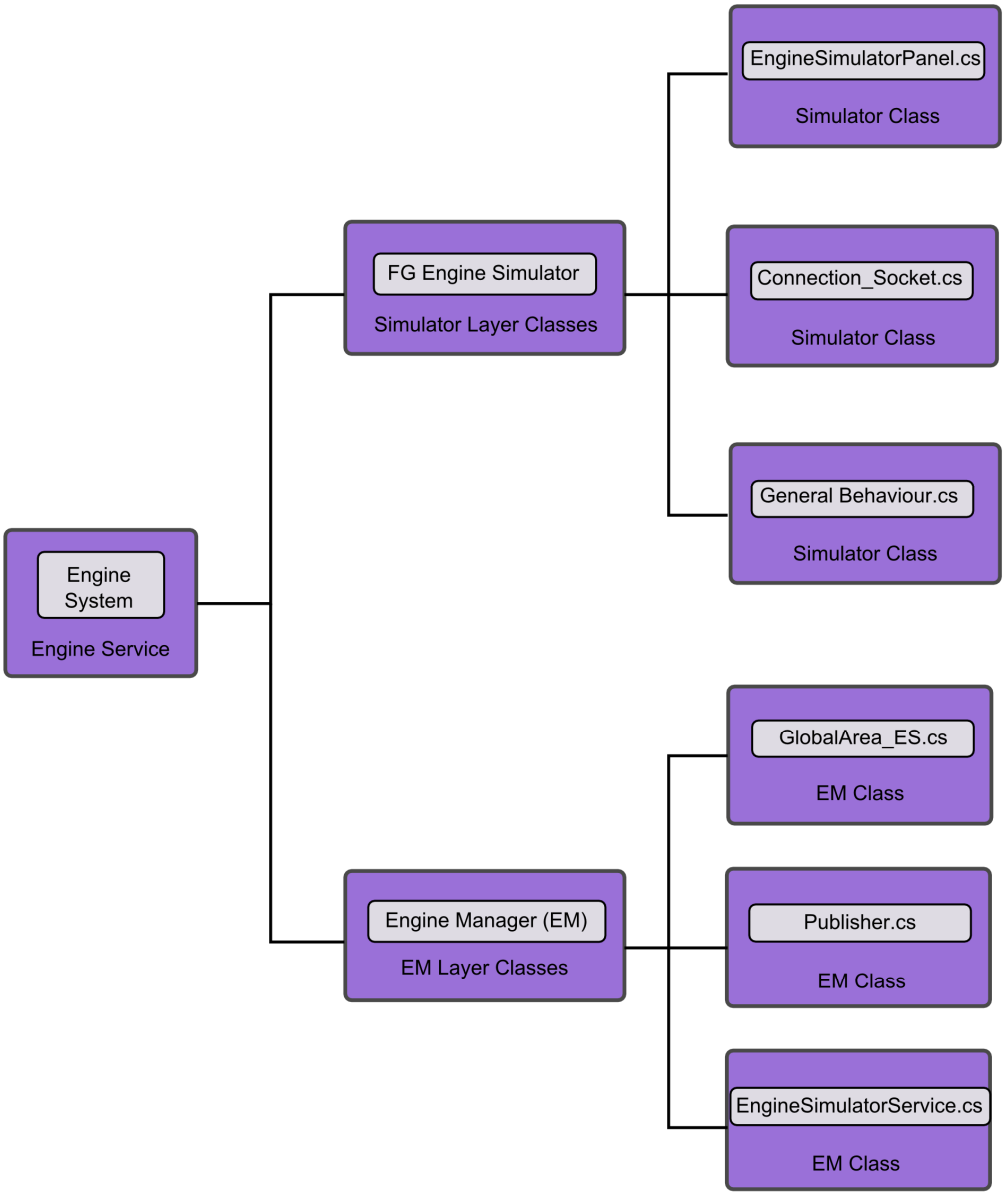

Fig.4.2 Engine Simulator Architecture.

As the previous simulator, the FG Engine Simulator layer will be replaced by sensors of the engine system of real UAS. For this reason, we divide the Engine System in two layers. The EM is the only layer that could reused in the future.

Design

We have divided the section into two groups for explanation of the design of the simulator. The first, called *code design* has the explanation in detail of each of the classes that have been used for programming the simulator code. The second, called *graphic interfaces design*, presents a general overview of the graphical design of the Engine Simulator.

Code Design

Next, we are going to explain more detail each of the classes of the Engine System.

The FG Engine Simulator has the following classes:

- **Connection_Socket.cs**: This class establishes the socket connection between the Engine Simulator and the FG. Is the same class that is explained in the Weather Simulator.

- **EngineSimulatorPanel.cs:** This class is the main form throughout the application. It is the main graphic interfaces of the simulator. With this class the user can interact with the Engine Simulator. In the *implementation* subsection there is a more detailed explanation of this class.

- **General Behaviour.cs:** This class allows to visualize all the graphics of the Engine Simulator in a new windows form.

The Engine Manager (EM) has the following classes:

- **EngineSimulatorService.cs:** This class determines all the variables and events published by the engine system. If it is necessary, the user would have the posibility to subscribe to other services with this class. The EngineSimulatorService.cs also is responsible for initialize the EngineSimulatorPanel.cs.

- **GlobalArea_ES.cs:** This class is like a box where there are all the values of variables and events to be publish. When the user simulates engine conditions, all parameters are stored in this new class. Thus, it will be take the values of the GlobalArea_ES.cs when it is necessary to publish data to the other services.

- **Publisher.cs:** This class is in charge of collecting the values of GlobalArea_ES.cs in order of publicating them on middleware. Thus, all services can use the parameters simulated by the Engine Simulator.

Graphic Interface Design

From the beginning, we knew that in order to make an efficient engine simulator, we needed to create a very intuitive service for the user. To achieve this, we need a simulator based in graphics and buttons.

All parameters of the Engine Simulator need a graph. Thus, the user sees clearly the evolution of parameter values. All graphs of the parameters in the same panel do not offer a good interpretation of what is happening.

To get a clear distribution of all parameters, we have based the design of the simulator on a tabs system. In each tab the user can see represented different engine parameter. In the last tab the user can access to the advanced options.

4.4 Implementation

We have designed a simulator with a panel of seven tabs. Six of the tabs are related to engine parameters and the last tab contains the advanced options. The panel is named Engine Simulator Panel.

The following explains in greater detail the features and the functionalities of each of the tabs. We represent the final implementation of Engine Simulator Panel in the Figure 4.3.

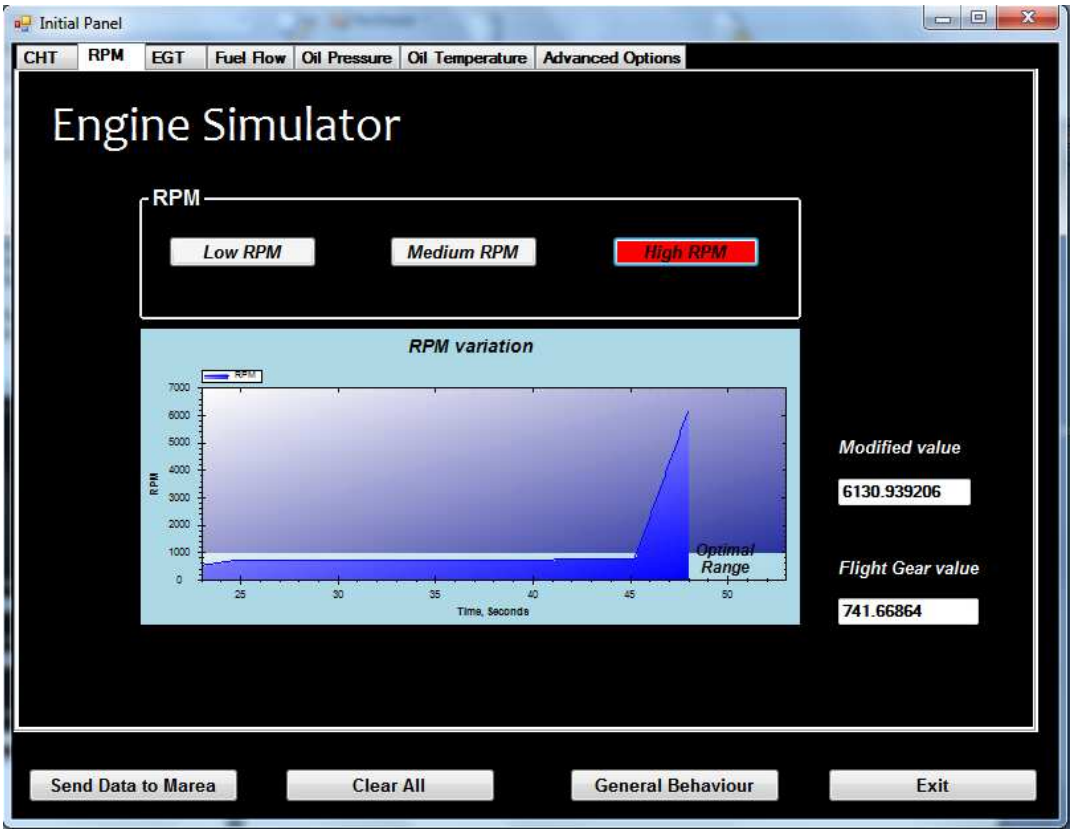

Fig.4.3 Engine Simulator Panel

Each tab of the Engine Simulator Panel belongs to a parameter that is simulated: CHT, EGT, RPM, oil pressure, oil temperature and fuel flow. Every tab is formed by their corresponding buttons.

Among all the tabs that are explained above we must emphasize one: the *fuel flow* tab is different from the others. In this tab we find a new box where we observe the value of the UAS tank. Also, the user has the possibility to simulate a failure of fuel tanks. With the Cause failure button the user simulates that the UAS does not have enough fuel.

For each of the parameters, the user has a group of 3 buttons. As it can be seen in Figure 4.3 the buttons are: *Low RPM* button, *Medium RPM* button and *High RPM* button, with this buttons the user can modify in greater or lesser degree the normal behavior of each parameter. This value increases or decreases progressively until it reaches its final value.

The value that is simulated will be consistently accounted on a graph. Each tab has its corresponding graph. We can see an example of graphics in the middle of the Figure 4.3. To plot the behavior of each parameter, we loaded into our project library, the dll of ZedGraph (www.zedgraph.org). It has allowed us to set up the visual appearance of the graphs and represent the evolution of each parameter.

All of the tabs share four general buttons: *Send Data to MAREA, Clear All, General Behaviour* and *Exit*. The first one is responsible for publishing the parameters simulated in the middleware. With the *Reset All* button, the user restarts the simulator parameters to their default values. With the *General Behaviour* button, the user can be seen the all graphs of the parameters in a new form. These buttons can be visualized at the bottom of Figure 4.3.

Our simulator publishes constantly to MAREA the instant parameters of the engine that receives of FG. If at any time any of the buttons of each tab is selected, pressing the *Send Data to MAREA* button, the parameter received of FG will be amended by a new value. This value depends on the button the user has selected. Thus, we can see the new simulated value in the box *changed value,* and the value received of FG in the box *FlightGear value*. The evolution of the value of each parameter can be seen at the graphic of each tab. The box *Changed value* and the box *FlightGear value* can be visualized to the right of the graph in Figure 4.3.

There is a relationship between each button and a maximum value and a minimum of that parameter of the engine. The maximum and minimum value can be modified in the advanced options in order to get a simulator more flexible and that it can adapts to different engine types.

Finally there is the Engine Simulator Panel. This tab is named Advanced Options (see Figure 4.4).

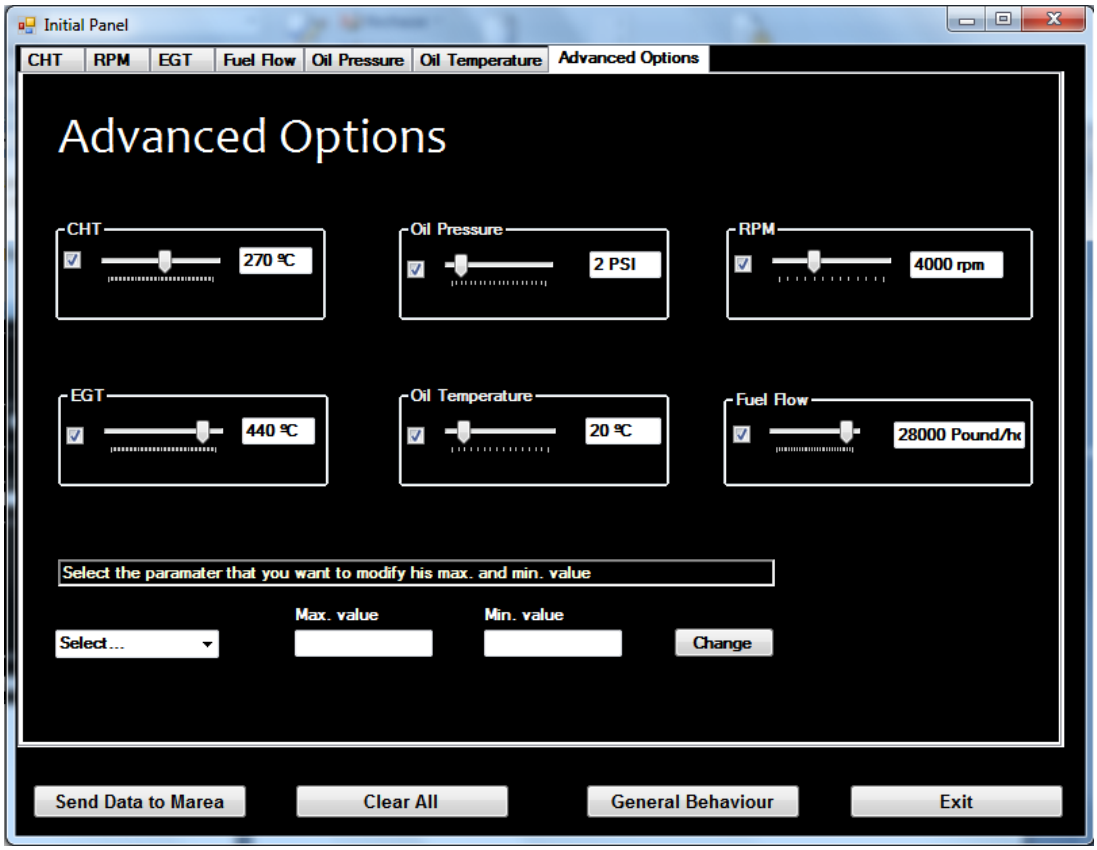

Fig.4.4 Advanced Options of Engine

The advanced options allow us to specify a determinate value of any parameter of the engine we want to modify. Thus, the user can simulate the engine with greater accuracy. As shown in Figure 4.4, this tab is divided into seven different sections. The first six refer to the engine parameters. Each one provides a specific value via a scrollbar.

The last section of this tab lets you select the maximum and minimum value among which oscillate every aspect configurable of the engine. Thus, the user can simulate any type of engine by introducing the maximum and minimum values for each parameter.

SECTION 5. ELECTRICAL SIMULATOR

In this section we are going to explain the all features of the Electrical Simulator and all the steps followed for its preparation.

This section starts with the Electrical Simulator introduction that explains the definition of the application and its objectives.

Next, it can be found the Electrical Simulator specification. It presents the functional goals of the Electrical Simulator and displays an overview of all the Electrical Simulator requirements.

This section follows with the architecture and design of the Electrical Simulator. It offers an explanation of the programming code and an outline of the structure of this program. It also presents an outline of the criteria for the design of the simulator.

The implementation of our project and the program results are depicted in the next subsection. This subsection details the implementation of the architecture and the design of the Electrical Simulator.

5.1 Introduction

The working of the electrical system of a UAS is decisive in a flight plan. A fault in the battery or in the alternator may cause irreparable damage in the UAS. The behavior of an aircraft may vary considerably if the working of the electrical system is not favorable.

At the ICARUS research group there is no service that considers the electrical system of a UAS. For this reason and in order to avoid all the problems explained in the previous paragraph, we need to create an Electrical Simulator.

The electrical simulator is an application that can emulate the electrical system of a UAS. The user has the option of adding to the UAV a number of devices and recreate situations of failure of these devices.

The goals of the electrical simulator are the following:

- Simulate an electrical system for the user to perform a study of the payload of the UAS.

- Publish all electrical parameters simulated to the rest of services of the ISIS platform. If some service of the platform needs some electrical parameter may subscribe without problem.

- Generate and publish alarms to the rest of services when dangerous situations are simulated on the UAS. Some services will need these alarms to perform their tasks and responses.

- Integrate Electrical Simulator in the ISIS platform. With the Electrical Simulator service, we increase the possibilities offered the services of the ISIS.

5.2 Specification

The Electrical Simulator must be a service that allows the user:

- Configure a large number of electrical devices. The devices that have to be taken into account are: video cameras, photo cameras, thermal cameras, gps, video transmitters, batteries, alternators, servos and autopilot.

- Activate or disable each of the simulated devices. Thus, the user can keep track of the consumption of the payload.

- View the remaining flight time with illustrations or graphics. Thus, the user can know if there is time to end the alleged mission of a UAS.

- Visualize how much energy is left for the battery. The user can control the energy level of the system.

- Visualize specific information to each device. The application must also show the consumption of each device. Thus, the user can determine which devices are useful at this time.

- Create an **easy** simulator and **simple usability**. Thus, the user can simulate an electrical system without being an expert in the field.

Finally, there is a graphic representation of the devices simulated by the Electrical Simulator. This representation can observe in Figure 5.1.

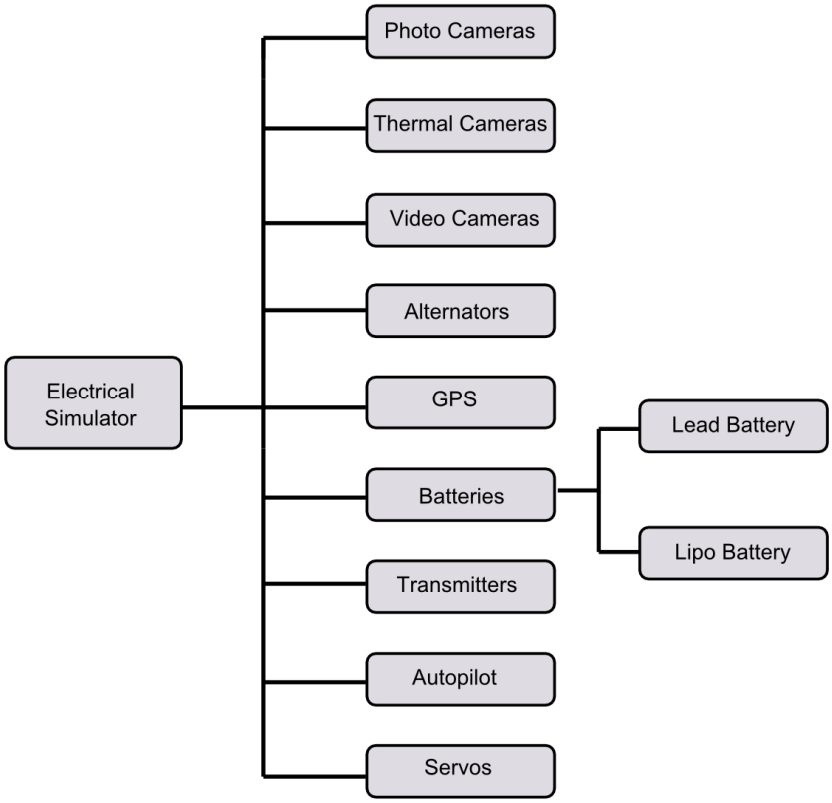

Fig.5.1 Parameters of Electrical Simulator.

5.3 Architecture and Design

The first part of the subsection will be referred to the program structure. It presents an overview of the architecture of the simulator.

The second part refers to all classes used for programming the simulator.

Architecture

Figure 5.2 shows the architecture of the programmed code of the Electrical System. It can be seen that the simulator is divided into two layers.

The first layer, called UAS Electrical Simulator, consists of all classes related to the graphical interface of the simulator. This layer is composed by the configuration forms of the simulator and by the information forms of the devices.

The second layer, called Electrical Manager (ELM), consists of classes that publish data to other services. All classes of the ELM will depend on a completely separate project called ElectricalSimulatorInterface. The interface is created to have a common definition of the classes. It is one way of sharing the definition of an object.

Specific definition of all classes that make up our project can be finding in the design subsection.

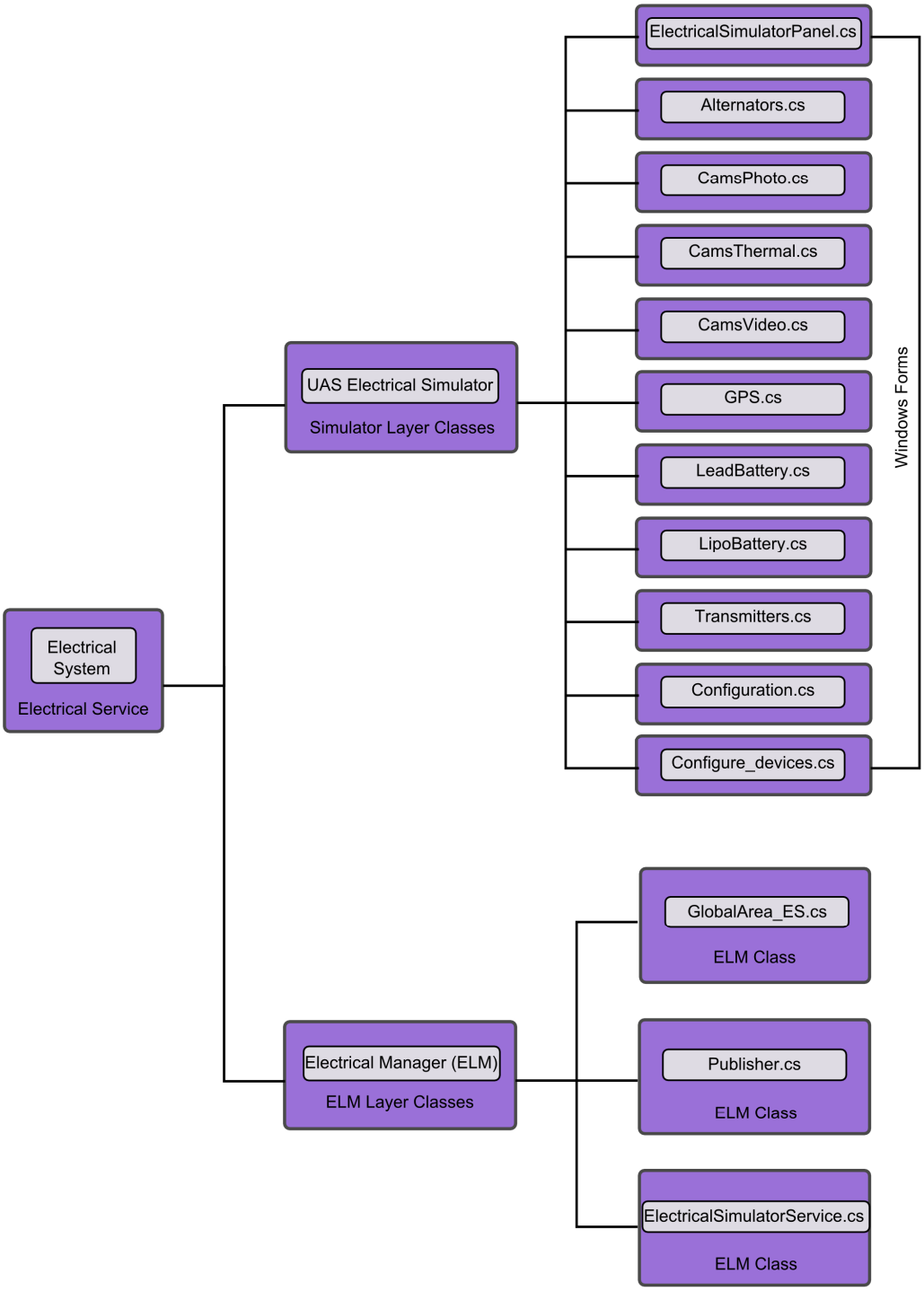

Fig.5.2 Electrical Simulator Architecture.

As the previous simulator, the UAS Electrical Simulator layer will be replaced by sensors of the electrical system of real UAS. For this reason, we divided the Electrical System in two layers. The ELM is the only layer that will be reused in the future.

Design

We have divided the section into two groups. The first, called *code design* is responsible for explaining in detail each of the classes that have been used for programming the simulator code. The second, called *graphic interfaces design*, presents a general overview of the graphical design of the Electrical Simulator.

Code Design

Next, we are going to explain more in detail each of the classes of the Electrical System.

The UAS Electrical Simulator has the following classes:

- **Configuration.cs:** This class is the first configuration panel that it initializes when the simulator is started. It allows us to configure how many devices we want to simulate. These devices are: photo cameras, video cameras, thermal cameras and batteries. The other devices, that cannot be set, are simulated by default when the user closes the configuration.cs class.

- **Configure_devices.cs:** The Electrical Simulator is programmed with specific models for each of the devices that are simulated. The application determines by default specific consumption for each device. This new class allows us to modify the devices configured by the simulator. Thus, the user will be able to modify patterns of the devices that he wants.

- **ElectricalSimulatorPanel.cs:** This class is the main form throughout the application. It is the main graphic interface of the simulator. With this class the user can interact with the Electrical Simulator. In the *implementation* subsection is a more detailed explanation of this class.

- **Classes of the devices:** UAS Electrical Simulator layer has eight classes dedicated to each of the devices. In each of these classes, the user can find the consumption of the device and detailed information of it.

The Electrical Manager (ELM) has the following classes:

- **ElectricalSimulatorService.cs:** This class determines all the variables and events published by the electrical system. In the case of being necessary, the user would have the posibility to subscribe to other services with this class. The ElectricalSimulatorService.cs also is responsible for initialize the ElectricalSimulatorPanel.cs.

- **GlobalArea_ES.cs:** This class stores all the values of variables and events to be publish. When the user simulates an electrical system, all

parameters are stored in this new class. Thus, it will be take the values of the GlobalArea_ES.cs when necessary to publish data to the other services.

- **Publisher.cs:** This class is responsible for collecting the values of GlobalArea_ES.cs and publish them to the middleware. Thus, all services can use the parameters simulated by the Electrical Simulator.

Graphic Interface Design

From the beginning, we knew that in order to design an efficient electrical simulator, we needed to create a very intuitive simulator for the user. The program must be able to collect the most important data on the same screen. So the user can see at any moment what is happening. For this reason, we decided to create a simulator based on graphs and illustrations.

Through a graph we can interpret on a fast and visual way how much estimated time we have flown. It is very important for the user to have always clear view of the status of the plane. For this reason we consider important the representation of a graph showing the time remaining before the batteries are discharged, with the devices currently enabled. This graphic is updated if a device is connected or disconnected.

It is also necessary that the user know at all times the consumption of each device. For this reason we have set graphs for each of these devices. Thus, the user may decide to consider whether or not activate a device. Also, we knew that the pictures in the main panel would help the user to understand the simulation. For this reason, we have included illustrations of each of the devices that are simulated. So, the user understands the simulation that is created.

The buttons are very useful tools for creating simulators that are quick and easy to use. The Electrical Simulator needs these buttons. They provide a lot of different utilities.

Finally, we consider that the representation of the charge value of the UAV is important. Monitoring the weight of the UAV is an important safety measure. Not all aircraft can withstand the same load. So the user must keep a tight check to not cause a loss of control of the plane. For all these reasons we decided to implement a TextBox in the main screen that is constantly showing the total payload of the aircraft. Thus we guarantee that the user can visualized this parameter.

5.4 Implementation

We have obtained a simulator with a main panel (Electrical Simulator Panel), a configuration panel (Configuration), eight panels concerning the devices installed on the UAV and an additional panel (Configure Devices) which allows us to change consumption and weight devices and batteries.

The following explains in greater detail the features and the functionalities of each of the forms.

Configuration

We represent the final design of Configuration panel of Electrical Simulator in the Figure 5.3.

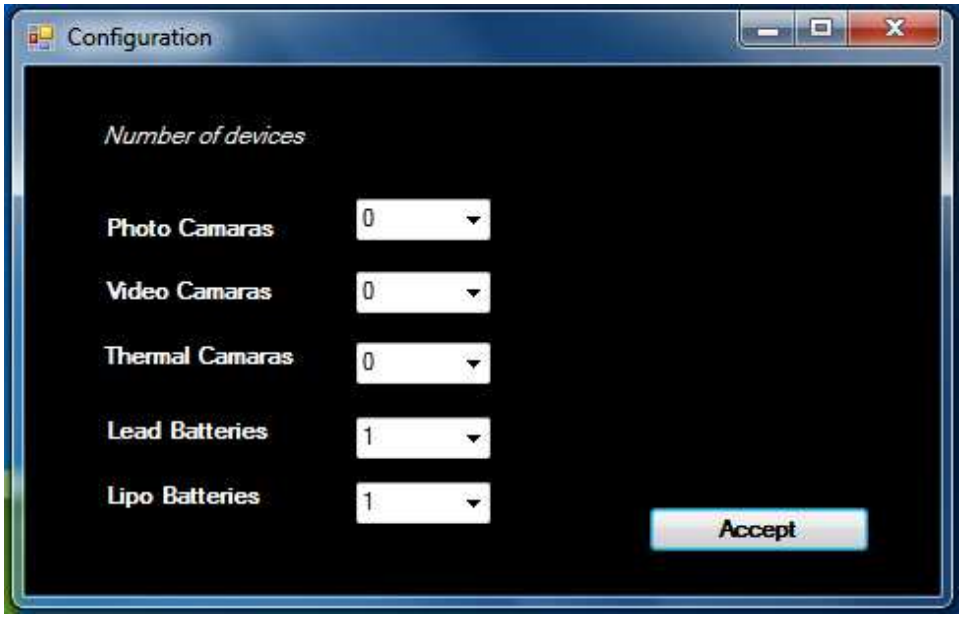

Fig.5.3 Configuration panel of electrical simulator.

The Configuration panel lets the user to select the desired quantity of devices to incorporate at the UAS. As shown in the Figure 5.3, the user can add to the payload: photo cameras, video cameras, thermal cameras, lead batteries and lipo batteries.

It will install five photo cameras, three video cameras, three thermal cameras and four batteries of both types. To choose the number of batteries possible we have relied on the document "*Engine and Fuel Manager System for Unmanned Aerial Vehicles"* being done by July Sagardoy Perez in the ICARUS group. On this document there is explained the implementation of the hardware of the plane and a proposal of a system capable of incorporating four batteries of each type.

Once the desired configuration devices were established, we will go to the main panel by clicking the *Accept* button.

Electrical Simulator Panel

We represent the final implementation of Electrical Simulator Panel in the Figure 5.4.

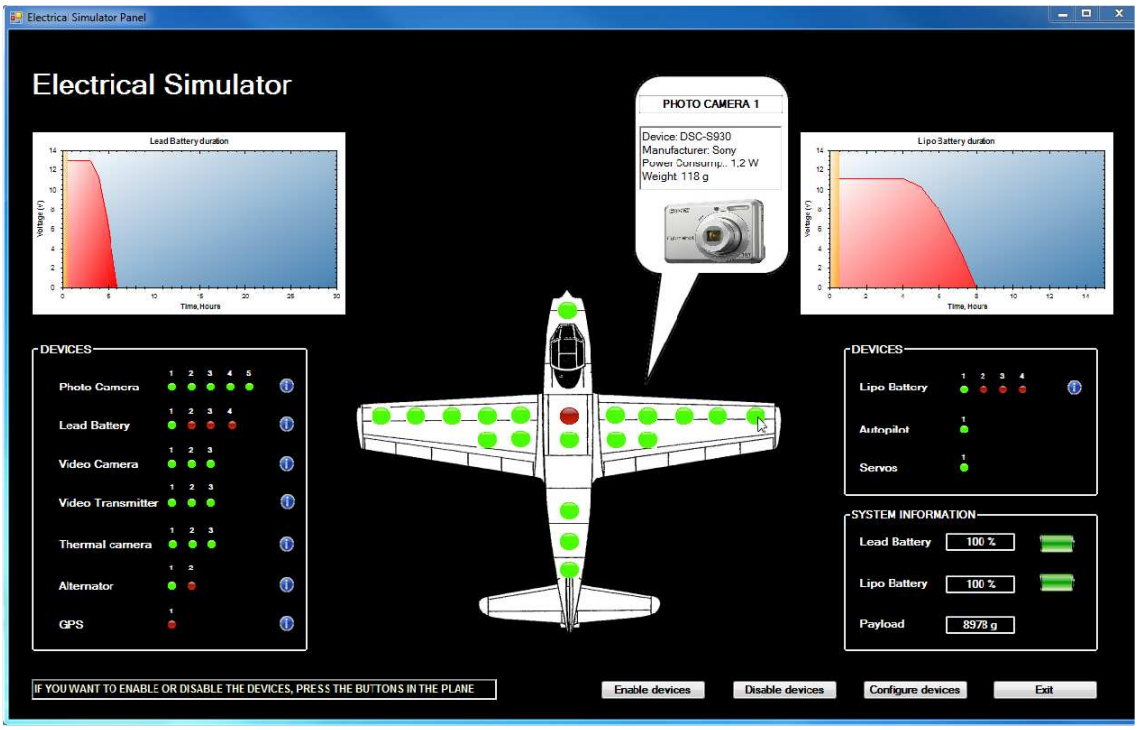

Fig.5.4 Electrical Simulator Panel.

The Electrical Simulator Panel is the main menu of our simulator and it is composed for five different sections. These five sections are: devices section, system information section, the plane section, graphical section and buttons section. Next, we are going to describe each electrical section from left to right of the Figure 5.4.

The first section that we are going to describe is the **devices** sections. The devices that can be incorporated into the UAVs are divided into two groups. The first group, to the left of the plane in Figure 5.4, is supplied from a lead battery. The second group, to the right of the plane in Figure 5.4, is supplied from Lipo batteries. In both sections the user can see which devices are activated (green light) or deactivated (red light). The user has the possibility to access to extra information about each device (see Figure 5.5) with the blue information button.

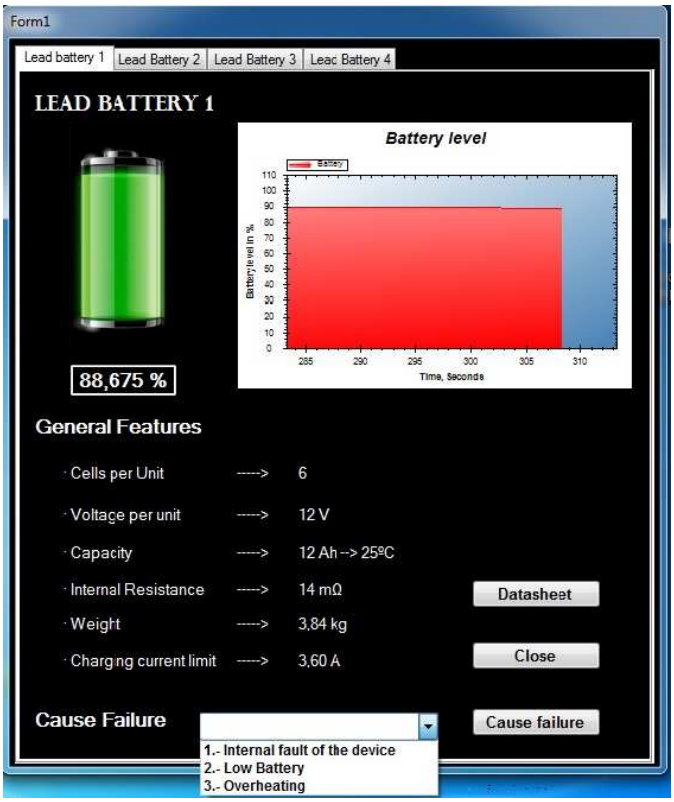

Fig.5.5 Devices information.

In this extra information, the user can see a picture and general characteristics of the device that is being simulated. Also, the user can see a graph showing the consumption of the device at all times.

Also, the user can simulate different types of device failures through a Combobox and a button. These are located at the bottom of Figure 5.5. The electrical simulator publishes an alarm in the middleware of this error with the button named above. Thus, the system of contingency knows that particular device has an error in its operation which, at that time, it is not operating.

With the info button on the video cameras the user also has the possibility to activate the record mode (*Rec* button) that consumes more current. With the info button on the photo cameras the user also has the possibility to activate the zoom mode and the photo mode. Both actions will produce an increase in consumption of the camera. These changes in consumption will be reflected in their corresponding graphs. The info button of lead batteries and Lipo batteries has a particularity. In both cases, we obtain a form with a representation of the energy that is left for each battery. Thus, users can take a visual check of the energy available for the flight.

Next, it can be seen the **graphic** sections at top of the *devices* sections in Figure 5.4. This section shows a visual representation of the estimated time that remains of flight. From the graph the user has the information of the volts delivered by the battery in function of time. Thus, the user ensures that during that time the system will work without the batteries run out completely.

As in the case of the engine simulator, these plots have been designed through the bookstore Zedgraph (see *http://www.zedgraph.org/*). The graph represents a vertical yellow line. This line shows the user the time it takes flight. This allows the user to identify where we are and how long it is estimated flight.

It can be seen the **plane** section to the right of the first *device* section in the Figure 5.4. This section is characterized for representing all the selected devices on a plane. Each device is represented by a green or red light on the silhouette of a UAS. The user can enable or disable all devices just by pressing the corresponding light. The user can get some little information on each device if the user holds your mouse over any of the lights of the plane.

The next section that we are going to describe is the **system information** section. It is located below the second *devices* section in Figure 5.4. In the system information section we can see two representations of the battery level and a display of the aircraft payload.

We see two levels of battery; the first is the lead battery and the second represents the remaining battery level of Lipo Battery. The representation of the payload of the UAS is directly dependent on the selected devices in the *Configuration* form (see Figure 5.3). The simulator adds to the variable *payload*, the weights of all embedded devices on the plane. Knowing the maximum load carried by the aircraft may determine whether the choice of devices is correct or not.

Finally, it can be seen the **buttons** section under the *system information* section in the Figure 5.4.In this final section are three different buttons: *enable devices, disable devices, configure devices* and *exit.*

With the first button, the user has the possibility to activate all the devices that are incorporated into the aircraft. With the second button, the user has the possibility of disabling all devices that are incorporated into the UAS. For security aspects, when the user presses the button to *disable devices,* batteries and alternators of the system are not deactivated. This ensures the safety of all system devices. *Autopilot* and *servo* devices also are not turned off with the disable devices button because are not configurable devices. With the configure devices button, we can see a supplementary form, which allows us to specify new models of devices or batteries (see Figure 5.6).

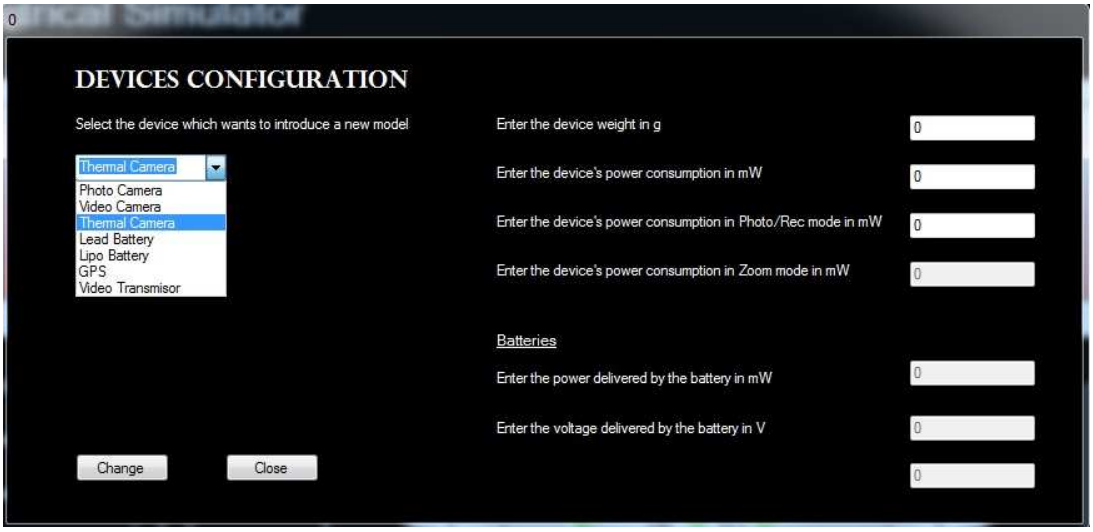

Fig.5.6 Configuration devices panel.

The user can replace batteries and devices installed by other different on the UAV. In order to configure this task, the user has to specify the weight and consumption of the new devices. Once the changes are done, the simulator modifies these parameters and emulates new elements. With this panel we obtain a simulator much more flexible because a small change the simulator is adapted to any type of device or battery.

SECTION 6. CONTINGENCY MANAGER

In this section we are going to explain the Contingency Manager (CM), all the features of the CM and the steps followed for its implementation. This section starts with the CM introduction which explains the CM definition and their objectives. Following we are going to describe the CM architecture, the CM design and implementation. To finish this section, we are going to present different use cases of the CM.

6.1 Introduction

The CM is the USAL service which centralizes all the alarms and contingencies of the system. The CM acquires and processes all the possible hazard situations to recover the correct status of the system. In order to design a robust system to manage UAS civil missions, contingency situations have to be taken in account. Any little failure of the system can achieve success of the mission in dangerous. All this warning, alarms, failures have to be treated to offer an intelligent response. The CM objectives are:

- To manage and centralize all the alarms and contingency situations.
- To find possible future UAS failures.
- To propose responses from contingency situation.

The CM is responsible for collecting status information related to multiple sources as: autopilot, engine, electrical, fuel, communications, etc. and identifying contingency situations. It is understood as contingency those situations which the UAS integrity is or will be in danger. If a contingency occurs, all involved services will be alerted and proper reaction will be taken according to the sort of contingency.

- Flight Contingencies: in case of weather changes we may force certain areas to be excluded from the operative flight plan. Other possible causes are that the expected performance of the UAS does not satisfy certain minimums or power sources do not provide the required levels of electrical energy, or fuel consumption does not behave as expected.
- Payload Contingencies: in case a given payload element fails some predefined actions need to be taken. If the payload element is critical for the flight, the flight plan needs to be terminated as soon as possible; if the contingency is critical for the mission, the mission is canceled or its objectives are reduced. If the contingency only affects the operation partially, the degraded conditions are annotated for further failures.

- Mission Contingencies: in case the expected mission results are not achieved due to any unexpected situation, mission objectives may be reduced or totally canceled.

- Awareness Contingencies: in case the airspace is not segregated another aircrafts can force flight plan changes or mission deviations.

6.2 Contingency Manager Architecture

The CM is composed by two pieces of software: the Health Monitor (HM) and the Contingency Intelligent Control (CIC). The first one is in charge of gathering and pre-processing all the information required to evaluate the UAS status. The second one provides the CM intelligence to evaluate all UAS pre-processed information and generate an intelligent response in front of any contingency.

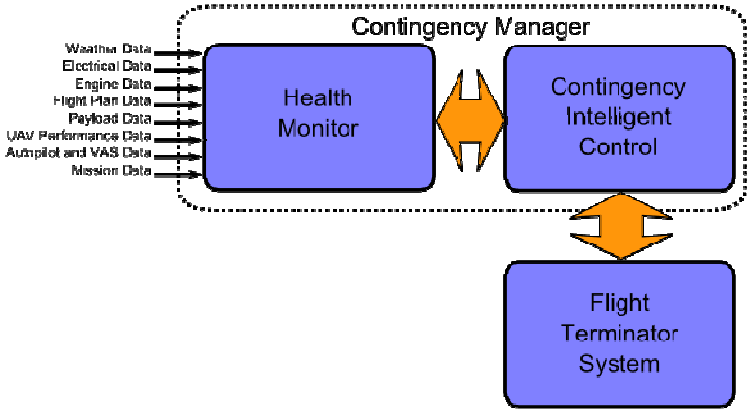

Fig. 6.1 Contingency Manager Architecture.

The HM process and gathers all the information needed to take a contingency decision. It is subscribed to the most relevant UAS information. This information is stored periodically, the CM information repository in order to be checked in that way to find any future contingency. To search UAS contingencies, the service will occasionally need mission or flight plan information. For example the mission time has to be compared with energy time or fuel time. This information will be achieved on demand to reduce network traffic. To sum up; the HM gathers all the information needed by the service to look for any contingency. This information can arrive periodically or on demand. When the HM finds a contingency, it is sent to the CIC in order to be classified.

The CIC gives the system intelligence and basically, it is in charge of responding or proposing different responses in front of any contingency preserving the UAS integrity. The CIC classifies the contingency in three categories: minor, hazardous and catastrophic. Each category has different responses as it is shown in the Figure 6.2.

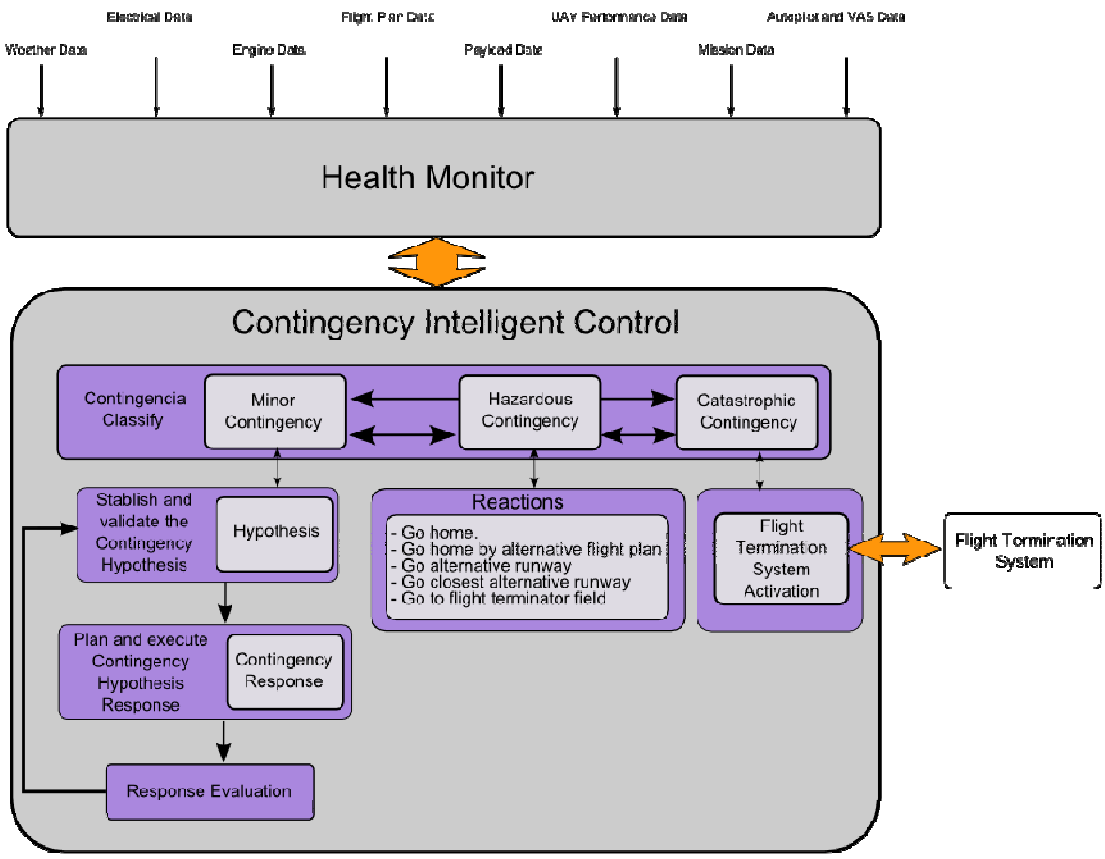

Fig.6.2 Contingency Intelligent Control Overview Architecture.

In the Figure 6.2 is depicted the contingency reactions for each category.

- The most important and restrictive category is the Catastrophic Contingency. The system enters in this state when the UAS cannot be recovered. Therefore, the mission has to be terminated ensuring enough safety. The Catastrophic Contingency module actives the Flight Termination System (FTS). The FTS commonly will be composed by parachute system to ensure the system safety and reduces ground crash risk.

- Next to the Catastrophic Contingency module it is found the Hazardous Contingency module. With this component we manage any contingency which interrupts or will interrupt the normal mission development. It is very important to prevent this type of contingencies because they might finish as a Catastrophic Contingency. On the other side; a proper and quick contingency detection can save the UAS platform.

- Finally the Minor Contingency component is shown in the left of the Figure 6.2 Minor Contingency treats any little anomaly or failure which can be recovery. This module establishes a contingency hypothesis and it plans and executes a response. After that the module monitor the system response until the contingency disappears.

As it is shown in Figure 6.2, the CM has a protocol to look the cause up and response in front of the Minor Contingencies. In order to response in a correct way the CIC is divided in three different pieces:

- Hypothesis: During this phase the CIC searches the cause of the alarm. The CIC studies all the information gathered by the Health Monitor. The result of this study is the hypothesis of the warning. Sometimes the CIC may find several hypotheses. In these cases the operator will have to choose the correct cause of the warning. The CIC, through a probabilistic method, will propose the origin of the failure. To sum up; in this phase we establish and validate the contingency hypothesis.

- Contingency Response: When is the contingency located, the CIC has to plan and execute the contingency hypothesis responses. This phase has pre-defined responses which have been pre-loaded during the dispatch process. The CIC can present different responses in priority order to the operator.

- Evaluation Response: When the operator has taken the decision, the CIC starts a new process to evaluate the response. This phase has to check that the contingency has terminated. In other case, and after a timeout, all the process has to be repeated again.

6.3 Design

In the Figure 6.3 we can see the different classes of the CM, which are divided in two different groups: HM Layer Classes and CIC Layer Classes.

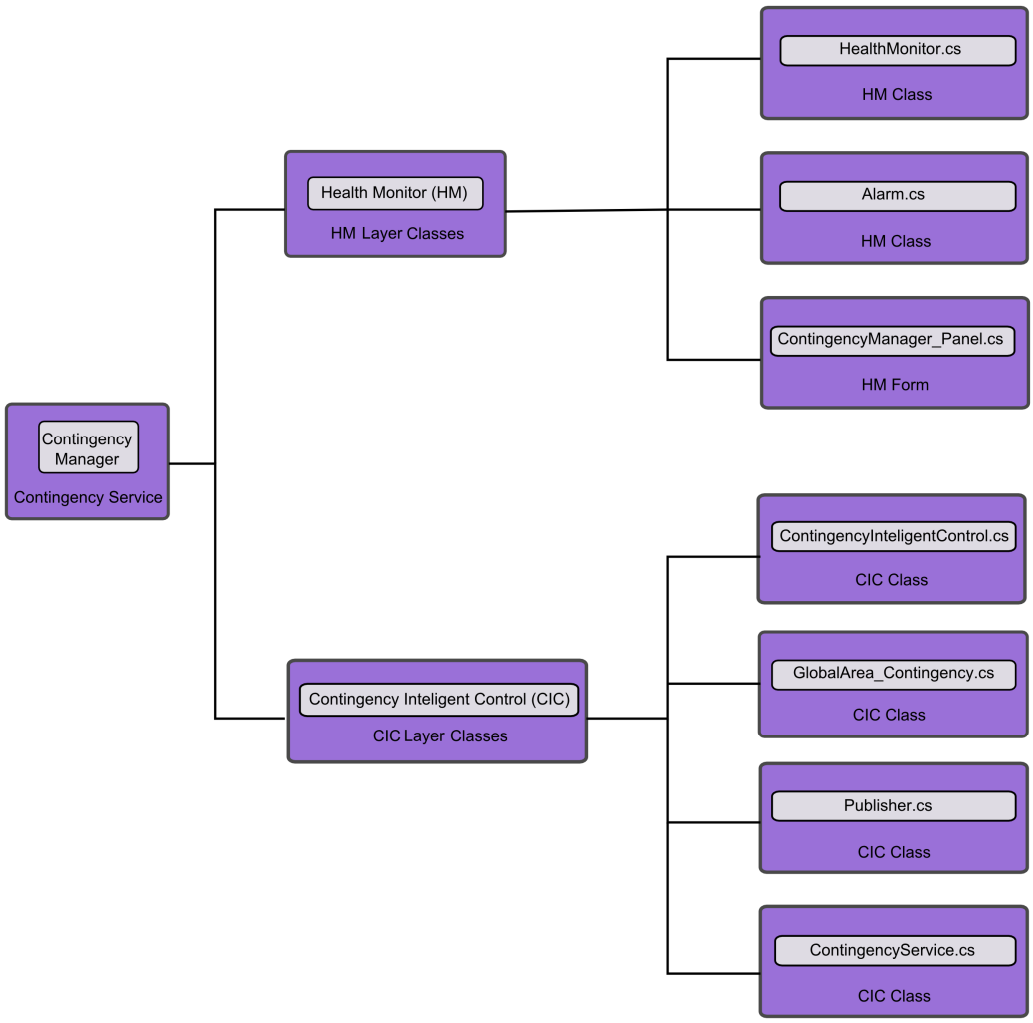

Fig.6.3 Contingency classes

Next, we are going to explain in more detail each of the classes of the CM:

- **ContingencyManager_Panel.cs:** This class is the main form throughout the application. It is the main graphic interfaces of the CM. With this class the user can see the alarms received by the HM. In the *implementation* subsection is a more detailed explanation of this class.

- **HealthMonitor.cs:** This class implements the HM, that we have explained previously.

- **ContingencyIntelligentControl.cs:** This class implements the CIC, that we have explained previously.

- **Alarm.cs:** This class define the structure of the object that the HM sends to the CIC.

- **ContingencyService.cs:** This class determines all the variables and events published by the CM. If it is necessary, the user would have the posibility to subscribe to other services with this class. The ContingencyService.cs also is responsible for initialize the ContingencyManager_Panel.cs.

- **GlobalArea_Contingency.cs:** This class is like a box where there are all the values of variables and events to be publish. When the CIC proposes a solution, all parameters are stored in this new class. Thus, it will be take the values of the GlobalArea_Contingency.cs when it is necessary to publish data to the other services.

- **Publisher.cs:** This class is responsible for collecting the values of GlobalArea_Contingency.cs for publication in middleware.

6.4 Implementation

The CM implementation is divided in three pieces of software, the HM, the CIC and the Contingency Manager Panel (CMP).

In the CMP (see Figure 6.4) we can see all the alarms received by the HM. When an alarm arrives to the HM, the CMP illuminate the button that corresponds to the type of alarm that the HM has received. The behavior of the HM and the CIC we have explained previously.

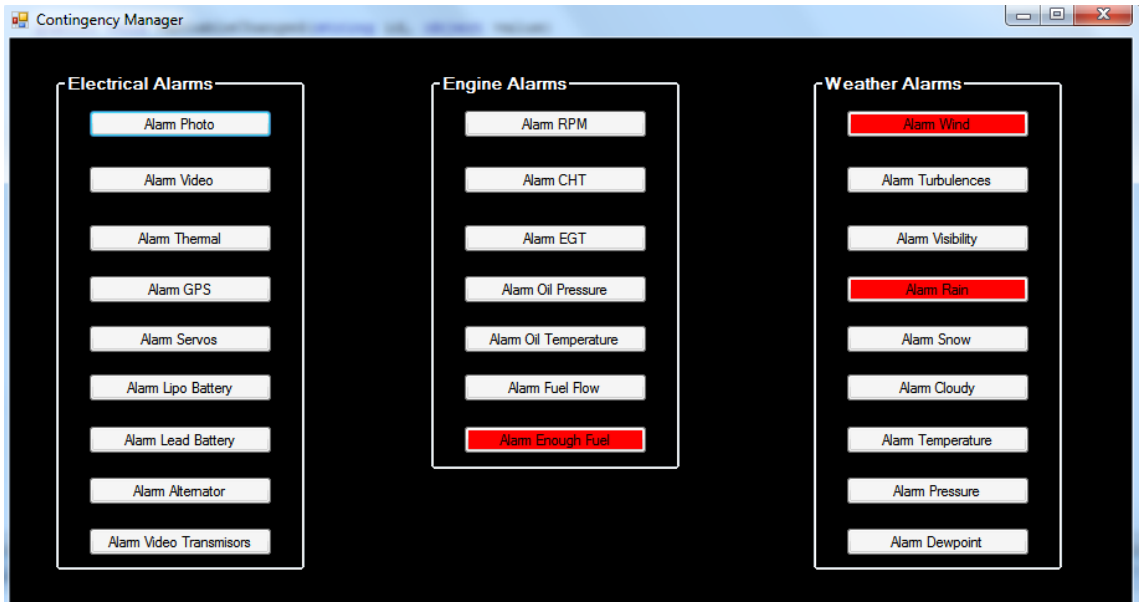

Fig.6.4 Contingency Manager Panel

The alarms are classified as follows:

- **Electrical Alarms**: collects the alarms caused by the devices of the electrical system of the UAS.
- **Engine Alarms**: collects the alarms caused by the engine of the UAS.
- **Weather Alarms**: collects the alarms caused by the weather effects that affect to the UAS behavior.

6.5 Contingency Manager Use Cases

This subsection describes the CM use cases from the different contingency areas. The objective of this section is to validate, through examples, the CM architecture. Emergency procedures need experience statistics in order to tune in the UAS responses, these responses are predefined during the dispatcher phase. Therefore, the CM will be continuously growing with the UAS flight experiences.

Weather Contingencies Use Case

In the Figure 6.5 we can see a wind contingency and how the HM and the CIC solves this problem. The CM solves more weather contingencies; we can see it in the section 7 of the appendix.

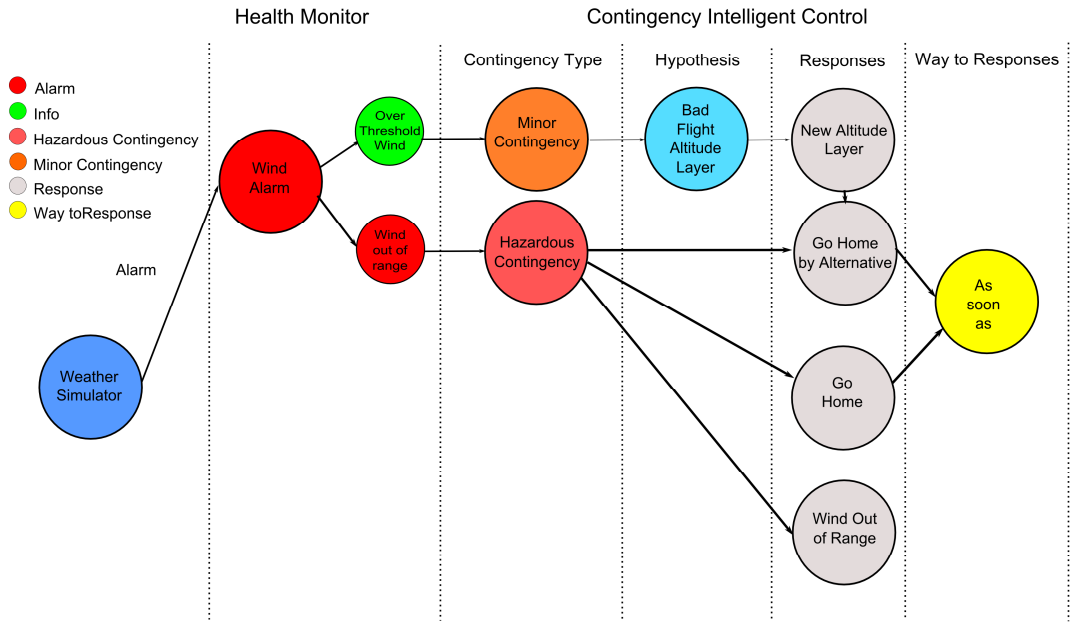

Fig.6.5 Weather use case

In this example the Weather Manager (WM) publishes a wind alarm which is detected by the HM of the CM. The HM classifies the alarm in a minor contingency or in a hazardous contingency. The CIC receives the classified contingency, in the case of the minor contingency it considers that the UAS has a bad flight altitude layer. Then the CIC proposes to the UAS to fly in a new altitude layer. If this solution isn't effective the CIC, proposes to Go Home by Alternative.
In the case of the hazardous contingency, the CIC proposes directly to Go Home by Alternative or Go Home as soon as possible.

The solutions that the CIC proposes have been predefined in the dispatcher phase.

Engine Contingencies Use Case

Next figure show a fuel contingency and how the HM and the CIC solves this problem. The CM solves more engine contingencies; we can see it in the section 7 of the appendix.

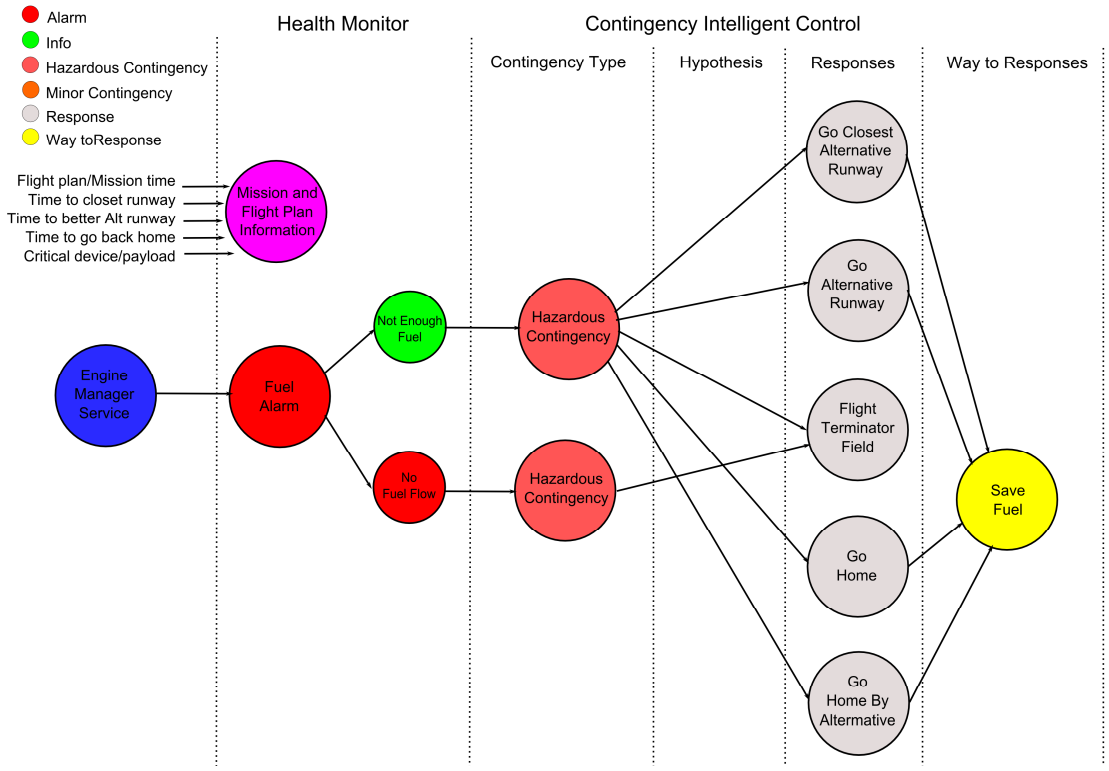

Fig.6.6 Engine use case

In this example the Engine Manager (EM) publishes a fuel alarm which is detected by the HM of the CM. The HM classifies the alarm in a not enough fuel or no fuel flow. Both alarms are classified like a hazardous contingency.

The CIC receives the classified contingency, in the case of the not enough fuel it proposes to the UAS to Go Home, Go Alternative Runway, Flight Terminator Field, Go Closet Alternative Runway or Go Home by Alternative depending on the time of fuel remaining.
In the case of no fuel flow, the CIC proposes directly Flight Terminator Field.

Electrical Contingencies Use Case

Next figure show an alternator contingency and how the HM and the CIC solves this problem. The CM solves more electrical contingencies; we can see it in the section 7 of the appendix.

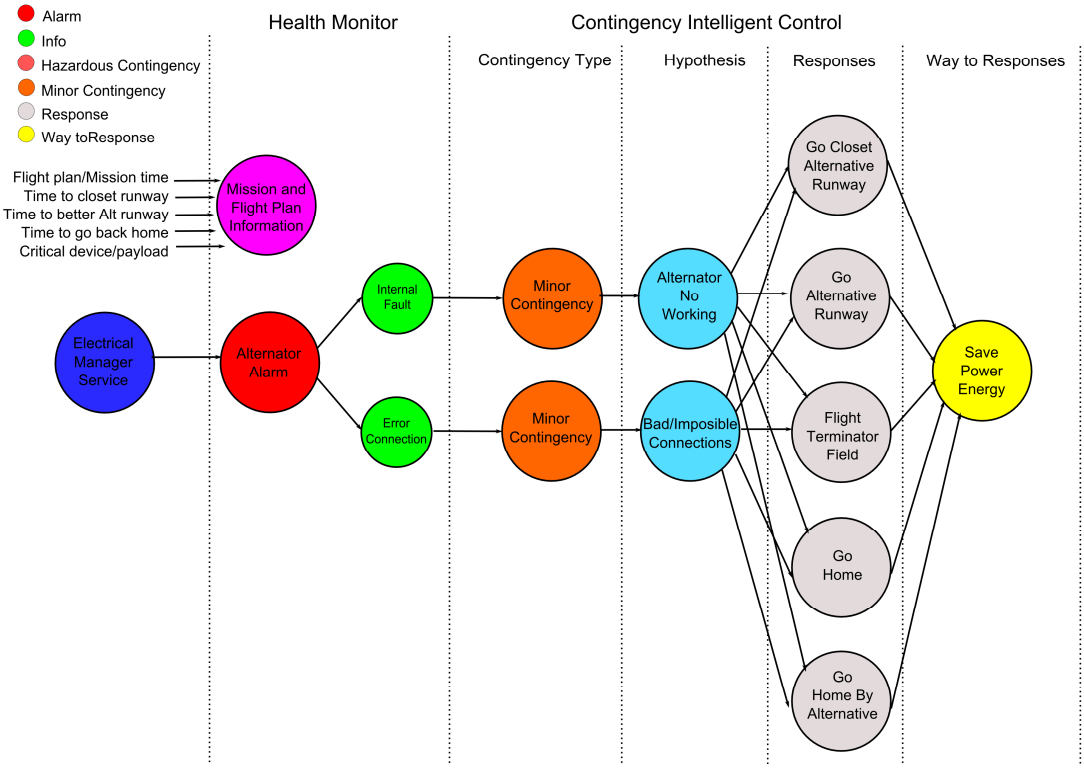

Fig.6.7 Electrical use case

In this example the Electrical Manager (ELM) publishes a fuel alarm which is detected by the HM of the CM. The HM classifies the alarm in an internal fault or error connection. Both alarms are classified like a minor contingency.
In both cases, the CIC proposes to the UAS to Go Home, Go Alternative Runway, Flight Terminator Field, Go Closet Alternative Runway or Go Home by Alternative depending on the time of fuel remaining and save power energy.
With this solution will turn off all the devices installed on the UAS that not affect to his normal behavior.

SECTION 7. FINAL BALANCE

In this section provides a final analysis of the project. The conclusions are the final review for the project. 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 CM. Finally there is an environmental impact description for that project.

7.1 Conclusions

The main objective of this project was to design, implement and integrate into the ISIS a Contingency Manager that can collect the UAS alarms, classify these contingencies and determine the action to be performed by the UAS.

For that the CM can act we had to design, implement and integrate into the ISIS, different simulators that are able to receive data from the components of a UAS.

Weather Conclusions

The Weather Simulator is able to simulate the parameters that give us the FG, with this application we can emulate different weather situations that are impossible to generate, without a simulator.

This application allows us to generate and publish all types of weather alarms in MAREA, for this reason we think that the Weather Simulator improves the ISIS platform.

Engine Conclusions

The Engine Simulator is able to simulate the engine system of a UAS, with this application we can emulate different engine failures that are impossible to generate in a laboratory, without a simulator.

The simulator allows us to generate and publish in MAREA, all types of engine alarms, thus the ISIS platform has been improved.

Electrical Conclusions

The Electrical Simulator is able to represent all the parameters needed to simulate an electrical system of a UAS; with this application we can emulate different failures in the electrical system of the UAS. These failures can’t be recreated in a laboratory because is not possible to repeat the action of turn on and turn off the electrical system, and generate failures of the different devices.

This application allows us to generate and publish all types of electrical alarms in MAREA, for this reason we think that the Electrical Simulator improves the ISIS platform.

Contingency Conclusions

The Contingency Manager centralizes all the alarms and contingencies of the system. The CM is an important piece of the USAL flight category that faces the complex problem of contingencies in simple and structured way.
It works with the Flight Plan Manager (FPM) to offer alternative responses in front of contingency situations. With these two pieces of software the UAS can manage hazardous situation, guaranteeing the UAS integrity. Also the service is capable to recover from minor contingencies and remain the UAS mission.

General Conclusions

The implementation of the simulators and the CM, was supposed to have been a relatively easy and quick workout. However, while we develop the different simulators and the CM, different problems arose. If we wanted to design and implement good simulators and a good CM, we had to take into account multitude of situations and problems that may arise during the flight of UAS. For this reason, the design and implementation of our project has not been so quick and easy as expected.

The integration of the simulators and the CM into the ISIS, have been done when we have finished the design and implementation of all simulators and the CM. It has been a difficult task because we didn't made the other services of the ISIS platform and we have needed time to understand how functions.

However, the general balance is excellent. A Contingency Manager has been designed, implemented and integrated into the ISIS. Also, the Weather Simulator, the Engine Simulator and the Electrical simulator have been designed, implemented and integrated into the ISIS.

7.2 Future lines of work

The three simulators and the Contingency Manager are beta versions, for this reason we think that all of these applications are improvable.

About the simulators can be implemented the following aspects:

- Collect the data of any type of internet service about the real weather conditions.

- Realize the simulators with another technology like WPF (Windows Presentation Foundation), thus the simulators will have a better visual aspect.

- Realize a connections system between the devices and the batteries of the Electrical Simulator. This system should be flexible and totally configurable for the user.

- The distribution of the Electrical Simulator devices in the UAS should be configurable for the user.

The future work of the CM will be addressed to improve the service intelligent response in front of hazardous and minor contingencies. All the contingencies have to be studied in order to offer several responses and help the operator in her decision. Another research line is to integrate and coordinate contingency responses with all the USAL services. A coordinated response will be more effective than a CM service response.

7.3 Environmental care

By the time the ICARUS UAS Platform gets ready to fly, its main application will be the detection and control of forest fires. Catalonia countryside is a hot-spot for these fires, as it has a warm, hot climate, especially in those summer months. ICARUS UAS Platform comes to replace the manned helicopters and airplanes that are currently used for fire-awareness purposes. These are high fuel consumers when compared to the fuel consumed by a small UAV engine, and its propellers are far noisier.

SECTION 8. BIBLIOGRAPHY

8.1 Books, articles and application notes

Royo, P., *"Contingency Manager, Functional Specifications"*, 23 pages, April 2009.

Tristancho, J., *"Flight Manual"*, 17 pages, May 2009.

Pastor, E.; Royo, P.; Santamaria, E.; Prats, X.; Barrado, C., *"In-Flight Contingency Management for Unmanned Aerial Vehicles"*, 15 pages.

Sagardoy, J., "*Engine and Fuel Manager System for Unmanned Aerial Vehicles"*, 75 pages, May 2009.

López, B., *"Ground Control Station for UAS over ICARUS System"*, 67 pages, September 2009.

López, J.; Royo, P.; Pastor, E.; Barrado, C.; Santamaria, E., *"A Middleware Architecture for Unmanned Aircraft Avionics"*, 20 pages.

Royo, P., *"An Open Architecture for the Integration of UAS Civil Applications"*, Computer Science PhD Thesis, 140-160 pages, May 2009.

8.2 WebPages

[1] Astromía: http://www.astromia.com/tierraluna/nubes.htm

[2] Zedgraph: http://www.zedgraph.org/

[3] Neoteo: http://www.neoteo.com/

[4] LaFlecha: http://www.laflecha.net/canales/ciencia

[5] Wikipedia: http://www.wikipedia.com

[6] Direct Industry: http://www.directindustry.es/

[7] Sutelco: http://www.sutelco.com/

[8] Ukai: http://www.ukai.com/

[9] Datasheet Catalog: http://www.datasheetcatalog.net/

[10] Seguridad Plus: http://www.seguridadplus.com

ANNEXOS

TÍTOL DEL TFC: Contingency Manager for ICARUS Simulated Integrated Scenario

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTORS: Oriol Caro Ignacio
Juanjo Rodríguez Carvajal

DIRECTORS: Pablo Royo Chic i Juan Manuel Lema Rosas

DATA: 9 de març de 2010

INDEX

SECTION 1. DATA TRANSMISSION PROTOCOLS

In this section, the basis of the transmission protocols will be studied. It is interesting to know how it performs the data transmission between the simulators and the FG.

1.1 Transmission Control Protocol (TCP)

The **TCP** is one of the fundamental protocols in the Internet. TCP is one of the two original components of the suite (the other being Internet Protocol, or IP), so the entire suite is commonly referred to as *TCP/IP*. Many programs within a data network comprised of computers can use TCP to establish *connections* between them through which you can send a data stream. The protocol guarantees that data will be delivered to its destination without errors and in the same order they were transmitted. It also provides a mechanism for distinguishing different applications within a single machine, through the concept of port. This protocol needs to establish a prior connection between sender and receiver before data transmission.

A TCP segment consists of a segment *header* and a *data* section. The TCP header contains 10 mandatory fields, and an optional extension field (*Options*, pink background in table). The data section follows the header. Its contents are the payload data carried for the application.

Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Source port																Destination port															
32	Sequence number																															
64	Acknowledgment number																															
96	Data offset				Reserved				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
128	Checksum																Urgent pointer															
160 ...	Options (if Data Offset > 5) ...																															

Fig.1.1 TCP segment structure.

1.2 User Datagram Protocol (UDP)

The **UDP** is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagram, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. UDP is sometimes called the Universal Datagram Protocol. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple transmission model without implicit hand-shaking dialogues for guaranteeing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagram may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games.

bits	0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

Fig.1.2 UDP packet structure.

1.3 Comparison of UDP and TCP

TCP is a connection-oriented protocol, which means that it requires handshaking to set up end-to-end communications. Once a connection is set up user data may be sent bi-directionally over the connection.

- **Reliable:** TCP manages message acknowledgment, retransmission and timeout. Multiple attempts to deliver the message are made. If it gets lost along the way, the server will re-request the lost part. In TCP, there's either no missing data, or, in case of multiple timeouts, the connection is dropped.

- **Ordered:** if two messages are sent over a connection in sequence, the first message will reach the receiving application first. When data

segments arrive in the wrong order, TCP buffers the out-of-order data until all data can be properly re-ordered and delivered to the application.

- **Heavyweight**: TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.

- **Streaming**: Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.

UDP is a simpler message-based connectionless protocol. Connectionless protocols do not set up a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver.

- **Unreliable**: When a message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission or timeout.

- **Not ordered**: If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.

- **Lightweight:** There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.

- **Datagram:** Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.

1.4 Protocols and Ports used in the transmissions

The UDP protocol allows our simulator to run without the need of having established a previous connection to the FG. That is why this protocol has been chosen for all the sockets in our applications.

The ports of the sockets in our applications are used with the 550X series following the FG transmission protocol. The flight simulator uses these ports to show all of its parameters through the network. Those used by the FG are: 5500 for https, 5501 for props and 5502 for jpg-httpd. The choice of our ports has been totally random.

Next, we are going to explain the ports that are used for the transmissions between simulators. The ports are:

- **5506:** This port is used for input socket. The input sockets are those that transmit information from the simulators created to the FG. This socket uses UDP protocol and it is used for sending information with the sea as

reference level. The 5506 port is used by the Weather Simulator to send the simulated data. Data transmission is accomplished via an XML file called Weather_Aloft.xml.

- **5507:** This port is used for output socket. The output sockets are those that transmit information from the FG to the simulators created. This socket uses UDP protocol and it is used for receiving the telemetry of the FG. The 5507 port is used by all simulators to send the simulated data. Data transmission is accomplished via an XML file called EngineTelemetry.xml.

- **5508:** This port is used for input socket. This socket uses UDP protocol and it is used for sending information with the ground as reference level. The 5508 port is used by the Weather Simulator to send the simulated data. Data transmission is accomplished via an XML file called Weather_Boundary.xml.

SECTION 2. TYPES OF CLOUDS

This section presents the different types of clouds that exist. With this study, the user can better understand what are the conditions that are simulated on the Weather Simulator.

Next, we are going to explain with more details each of types of clouds:

- **Cumulus**: clouds of vertical development. They have great size with a massive appearance of shadows when they are very marked between the Sun and the observer, namely, they are gray clouds. The clusters are for the good weather in low humidity and little vertical air movement. They can become large rise reaching intense storms and downpours.

Fig.2.1 Cumulus.

- **Stratus**: stratified clouds. They look like a gray fog bank without being able to observe a definite structure or regular. They have patches of different degrees of opacity and color variations of gray. During the fall and winter stratus can stay in the sky throughout the day giving a sad look at the sky.

Fig.2.2 Stratus.

- **Nimbostratus**: capable of forming rain clouds. They look like a regular layer of dark gray with varying degrees of opacity. Not infrequently one can see a slightly striated appearance which corresponds to different degrees of opacity and color variations of gray. Clouds are typical of spring and summer rain and snow during winter.

Fig.2.3 Nimbostratus.

- **Cirrus**: clouds of ice crystals. They are white clouds, transparent and without internal shadows present an appearance of long thin filaments. These filaments may have an even distribution in the form of parallel lines, either straight or sinuous. The overall appearance is as if the sky had been covered by the brush strokes.

Fig.2.4 Cirrus.

We found 16 different types if we make a specific study of clouds.

SECTION 3. ATMOSPHERIC LAYERS

In this section, the basis of the atmospheric layers will be studied. It is interesting to know why the FG simulates weather conditions with five layers.

We are going to explain how many layers the atmosphere has and what its functions are.

The altitudes of the FG represent the seven layers that divide the Earth's atmosphere (see Figure 3.1). Each layer has a particular function:

- **Troposphere:** The first one, called the troposphere, produces weather events like rain and wind.
- **Stratosphere:** The second one, called the stratosphere, absorbs harmful shortwave radiation through the ozone.
- **Mesosphere:** The third one, called the mesosphere, is important for the ionization and chemical reactions that occur in it.
- **Thermosphere:** In the fourth one, called the thermosphere, the air is very dim and the temperature changes with solar activity.
- **Ionosphere:** The fifth one, called the ionosphere, causes the phenomenon of the aurora and reflects long-wave radio waves.
- **Exosphere:** The sixth, called the exosphere, is the outer limit of the atmosphere.
- **Magnetosphere:** The last one, called the magnetosphere, is where the planet's magnetic field dominates the interplanetary magnetic field environment.

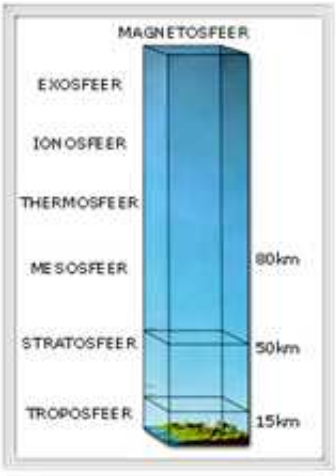

Fig.3.1 Atmospheric layers.

SECTION 4. CESSNA 172

This section presents the characteristics of a CESSNA 172 engine.

The values that Engine Simulator simulates refer to the engine of a Cessna 172. We have simulated this engine because it is a similar engine to the aircraft uses that is being built at the ICARUS group.

The Cessna 172 started life as a tricycle landing gear variant of the tail dragger Cessna 170, with a basic level of standard equipment. The first flight of the prototype was in November 1955. The 172 became an overnight sales success and over 1.400 were built in 1956, its first full year of production.

Early 172S were similar in appearance to the 170, with the same straight aft fuselage and tall gear legs, although the 172 had a straight vertical tail while the 170 had a rounded fin and rudder. Later 172 versions incorporated revised landing gear and the tail sweptback which is still in use today. The final aesthetic development in the mid-1960s was a lowered rear deck that allowed an aft window. Cessna advertised this added rear visibility as "Omni-Vision". This airframe configuration has Remained almost unchanged since then, except for updates in avionics and engines, including the Garmin G1000 glass cockpit in 2005. Production had been halted in the mid-1980s, but was Resumed in 1996 with the 160 hp (120 kW) **Cessna 172R Skyhawk** and was supplemented in 1998 by the 180 hp (135 kW) **Cessna 172S Skyhawk SP.**

Below are two tables which represent the general characteristics of this aircraft and its performance.

Performance	Values
Never exceed speed	163 knots (187 mph, 302 km/h)
Maximum speed	123 knots (141 mph, 228 km/h) at sea level
Cruise speed	122 knots (140mph, 226 km/h)
Range	610 nm (790 mi, 1,272 km) at 55% power at 12,000 ft (3,040 m)
Service ceiling	13,500 ft (4,116 m)
Rate of climb	720 ft/min (3.7 m/s)

Table 4.1 Performance of Cessna 172

General Characteristics	Values
Crew	1
Capacity	3 passengers
Length	27 ft 2 in (8.28 m)
Wingspan	36 ft 1 in (11.0 m)
Height	8 ft 11 in (2.72 m)
Wing area	174 ft² (16.2 m²)
Airfoil	NACA 2412 (modified)
Empty weight	1,620 lb (736 kg)
Useful load	830 lb (376 kg)
Max takeoff weight	2,450 lb (1,113 kg)
Powerplant	1× Lycoming IO-360-L2A flat-4 engine, 160 hp (120 kW) at 2,400 rpm
Zero-lift drag coefficient	0.0319
Drag area	5.58 ft² (0.52 m²)
Aspect ratio	7.32
Lift-to-drag ratio	11.6
Wing loading	14.1 lb/ft² (68.8 kg/m²)
Power/mass	15.3 lb/hp (9.25 kg/kW)

Table 4.2 General Characteristics of Cessna 172

SECTION 5. ENGINE SIMULATOR PARAMETERS

This section explains the function of all that the user will see in the Engine Simulator Panel. It will be explained each of the values that we have in our Engine Simulator.

CHT*:* Indicates the temperature of all cylinder heads or on a single CHT system, the hottest head. A Cylinder Head Temperature Gauge has a much shorter response time than the oil temperature gauge, so it can alert the pilot to issue a developing cooling more quickly. Engine overheating may be caused by:

- Running too long at a high power setting.
- Poor leaning technique
- Restricting the volume of cooling airflow too much.
- Insufficient delivery of lubricating oil to the engine's moving parts.

EGT*:* Indicates the temperature of the exhaust gas just after combustion. Used to set the fuel / air mixture (leaning) correctly.

CHT and EGT are measured in ° C and we can see the values between they moves in the table (see Table 5.1).

	Range			**Alarms**			**Unit**
	Min	Typ	Max	Warning	Severe	Critical	
CHT (Cylinder Head Temperature)	80	90	100	>100 <80	>300	>400	ºC
EGT (Exhaust Gas Temperature)	80	90	100	>100 <80	>300	>400	ºC

Table 5.1 CHT and EGT values

RPM*:* Is a unit of frequency of rotation: the number of full rotations completed in one minute around a fixed axis. It is used as a measure of rotational speed of a mechanical component.

Standards organizations generally recommend the symbol **r/min**, which is more consistent with the general use of unit symbols. This is not enforced as an international standard.

The corresponding unit in the International System of Units (SI) is hertz (symbol Hz) or s^{-1} (1/second). Revolutions per minute are converted to hertz through division by 60. Conversion from hertz to RPM is by multiplication with 60.

(5.1)

$$1rpm = 1 \cdot \left(\frac{r}{min}\right) = \left(\frac{1}{60}\right) Hz$$

Another related unit is the SI unit for angular velocity, radian per second ($rad \cdot s^{-1}$):

(5.2)

$$1rpm = 1 \cdot \left(\frac{r}{min}\right) = 2\pi \cdot \left(\frac{rad}{min}\right) = \left(\frac{2\pi}{60}\right) \cdot \left(\frac{rad}{s}\right) = 0.10471976 \left(\frac{rad}{s}\right)$$

In the Table 5.2 we can see the values between it moves.

	Min	Typ	Max	Warning	Severe	Critical	Unit
RPM	1000	-	7000	<1000 (depends on whether it is in the air, if it is no alarm)	>7000	>12000 <100 (<100: depends on whether it is in the air, if it is no alarm)	min-1

Table 5.2 RPM values

OLP *(Oil Pressure):* Indicates the supply pressure of the engine lubricant. Oil pressure is measured in PSI *(Pounds per Square Inch),* a unit whose value is equal to 1 pounds per inch square. Thus, the system of equivalents with respect to the international system would be as follows:

(5.3)

1 pound / square inch (psi) = 6894.75 Pascal

OLT *(Oil Temperature):* Indicates the engine oil temperature and is measured in ^{o}C.

FL *(Fuel Flow):* It indicates the fuel consumption per unit of time. In this case, it is used Pounds per hour as the measurement unit. Its equivalence with the international system is:

(5.4)

1 pound per hour [lb / h] = 0.000125998 kilogram per second [kg / s]

We can see the values in the Table 5.3.

	Min	Typ	Max	Warning	Severe	Critical	Unit
Oil pressure	10	15	20	<20	<15	<10	PSI
Oil Temperature	Ambient	50	150	>40	>60	>100	ºC
Fuel flow	0	-	16000	>16000	>18000	>20000	Pound/hour

Table 5.3 Oil pressure, oil temperature and fuel flow values.

SECTION 6. ELECTRICAL SIMULATOR DEVICES

This section explains what devices are used in the Electrical Simulator.

6.1 Electrical devices

First, we consider the most important parameters that can be found in the electrical system of a UAV to represent them in the main panel of the simulator.

We decided to divide the electrical system in two groups.

- The first group consists of all electrical devices of acquisition and data emission, i.e., digital cameras, video cameras, thermal cameras, video transmitters and GPS.

- The second group is the autopilot and all servos that allow the UAV can fly.

We have considered interesting that the user can manually enable and disable any device, thereby achieving generate emergencies.

Then, it can be see the devices that have been used.

Photo Cameras

Have been installed cameras, as it is interesting to take pictures at certain times in a mission. For example, it would be interesting to take a picture of the situation from an objective of the mission or any problem encountered during flight.

The camera that we used for the simulation is as follows, this camera allows us to take images with great accuracy and his weight is quite low, which helps us not to increase the payload of the UAV:

Fig.6.1 Sony DSC-S930.

Parameters	Values
Field of view	75º-32º (diag)
Focal length	28-75 mm
F-number	1-2.8
Resolution	4000 x 2656 pixel
Frame rate	3 fps

Table 6.1 Specifications of photo camera

Video Cameras

It has been implemented video cameras in the UAV, as it is interesting that we can make recordings of situations that can be seen from the plane.

As visual camera, we used the Lumenera Le11059c 11 Megapixel network camera with Tamron A09 zoom lens. Table A summarizes the relevant specifications of the cameras.

Fig.6.2 Lumenera Le11059c.

Parameters	Values
Field of view	75º-32º (diag)
Focal length	28-75 mm
F-number	1-2.8
Resolution	4000 x 2656 pixel
Frame rate	3 fps

Table 6.2 Specifications of video camera.

Thermal Cameras

Have been implemented thermal cameras in the UAV, as it is interesting to detect points where the temperature is higher. For example, points where it's starting a fire. If we know these points, we can prevent the fire.

As thermal camera, we used the FLIR A320 camera, a radiometric thermal camera working in the wavelength range from 7.5 o 13.0 µm and in the thermal range from 0 to 350ºC. It provides 320 x 240 pixel images of 32 bit floating point absolute temperature values.

Fig.6.3 ThermoVision A320.

Parameters	Values
Spectral range	7.5 to 13 µm
Temp. range	0 to 350ºC (±2ºC)
Field of view	25.0º x 18.8º
Focal length	18 mm
F-number	1.3
Resolution	320 x 240 pixel
Frame rate	9 fps

Table 6.3 Specifications of thermal camera.

Video Transmitters

The transmitters can receive audio and video images captured by the cameras of our plane. This is the best way to send the signal wirelessly and with all the quality offered by the 2.4 GHz band.

The transmitters that we used for the simulation is as follows:

Fig.6.4 Video Transmitters.

Parameters	Values
Frequency Band	2400MHz
Available Channel	8 Ch
Consumption Current	500 mA
Output Power	1000 mW
Power Supply	12V / 500mA
Weight	20 g

Table 6.4 Specifications of transmitters.

GPS

The Global Positioning System (GPS) (although its correct name is NAVSTAR-GPS) is a global navigation satellite system (GNSS) which allows knowing the position of a moving object through the reception of signals from a satellite network. GPS works through a network of 27 satellites (24 operational and 3 back) in orbit above the earth, at 20,200 km, with trajectories synchronized to cover the entire surface of the Earth.

The gps that we used for the simulation is as follows:

Fig.6.5 Mini Gps.

Parameters	Values
Receptor	66 Channels
Sensitivity	-159 dBm
Exact position	2,5 - 3,1 meters
Voltage	3,3 V
Consumption	32 mA
Weight	13 g

Table 6.5 Specifications of gps.

Autopilot

An autopilot is a mechanical, electrical, or hydraulic system used to guide a vehicle without assistance from a human being.

Servos

A servo motor is basically a mechanical actuator based on an engine and a set of gears that can multiply the torque of the final system, which has control elements to consistently monitor the position of a mechanical element that will be the liaison with the outside world.

Fig.6.6 Servomotor.

6.2 Batteries

Battery or accumulator is the device that stores electrical energy using electrochemical methods and that subsequently returns almost in its entirety. This cycle can be repeated by a specified number of times. This is a secondary electric generator, is a generator that cannot operate without being supplied electricity previously through what is called charging.

The battery management is essential in a flight simulator. It always needs to have a simulation of the remaining battery level and stimulation of flight time remaining to us. Battery depletion would produce irreparable damage to the aircraft. For all this, we consider that the representation of the aircraft battery is essential on the main screen of the simulator.

We determine the batteries we are representing at the Electrical Simulator with the help of all members of the ICARUS project.

For the first group of devices previously established we use a lead battery(GP 12120), because it is a higher capacity battery, and can feed a larger number of devices. If the battery is being finished may be charged by the alternator.

For the second group we use a Lipo battery (V-MAXX 35c), because it is a smaller capacity battery, and not have to feed so many devices. Moreover, this battery will be charged from the ground.

Lead Battery

Fig.6.7 *GP 12120*.

Parameters	Values
Cells per Unit	6
Voltage per unit	12 V
Capacity	12 Ah → 25ºC
Internal Resistance	14 mΩ
Weight	3,84 kg
Charging current limit	3,60 A

Table 6.6 Specifications of lead battery.

The lead-acid battery is a type of electrical energy accumulator, which consists of lead plates alternating with lead dioxide, which are separated by an element in dilute sulfuric acid soaked in distilled water, called an electrolyte. Each pair of these plates generates a voltage of 2 volts. As the typical configuration is 12 volts, is required above 6 pairs of plates to achieve the voltage of 12 volts. Depending on the total area of the plates, the battery will reach a certain capacity, whose unit is Ampere-Hour (Ah).

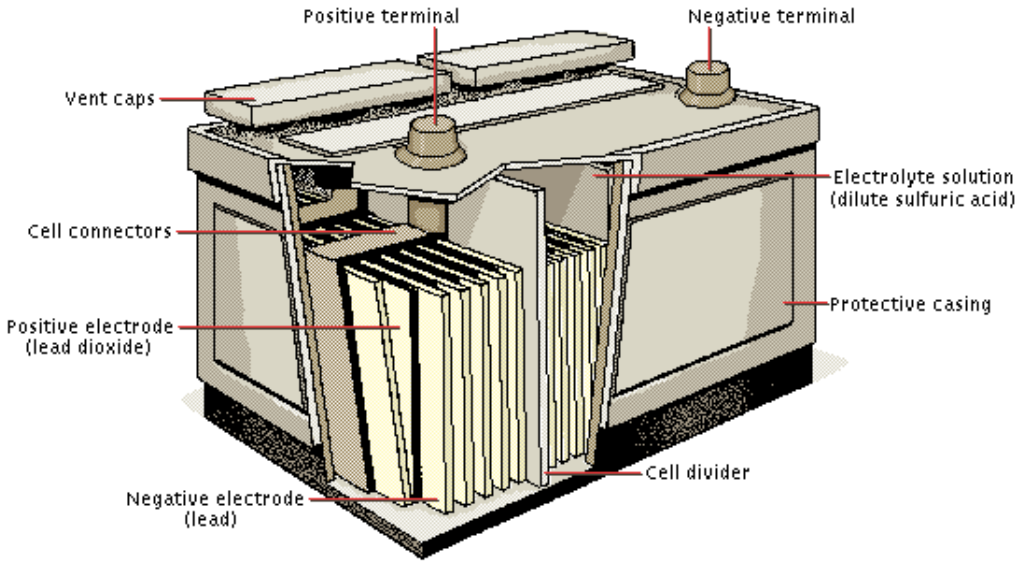

Fig.6.8 Lead Battery.

This type of battery will be able to recharge in flight due to an alternator. This ensures that all devices are always working.

Lipo Battery

Parameters	Values
Cells per Unit	6
Voltage per unit	12 V
Capacity	12 Ah → 25ºC
Internal Resistance	14 mΩ
Weight	3,84 kg
Charging current limit	3,60 A

Fig.6.9 V-MAXX 35c. **Table 6.7** Specifications of lipo battery.

Lipo batteries are composed of lithium and polymer, which gives them look "soft" and somewhat ungainly, though rather thin and lightweight. These batteries have a smaller size compared to the lead battery. Its size and weight make them very useful for small teams that require strength and durability.

Such batteries are a variation of the lithium-ion batteries (Li-ion). Its characteristics are very similar, but allow a higher energy density and a significantly higher discharge rate.

Normally these batteries are used to fuel helicopters, radio controlled planes and cars as they are able to deliver high doses of power consumption in large schemes.

In the simulation of the electrical system, these batteries are charged from the ground and will not be recharged during the flight. We will need to know how many batteries are needed to feed all the devices of the second group.

SECTION 7. CONTINGENCY MANAGER USE CASES

This section describes more deeply the CM use cases from the different contingency areas. The objective of this section is to validate, through examples, the CM architecture. Emergency procedures need experience statistics in order to tune in the UAS responses. Therefore, the CM will be continuously growing with the UAS flight experiences.

Weather Use Cases

Weather is an important factor to achieve mission success. Each aircraft has different flight performances. So weather conditions affect the each airframe in a different way. The weather conditions are changeable during the mission flight plan. In dispatcher phase we know the weather forecast for the flight plan. However, what happened if these weather conditions change or just if the weather forecast is not enough suitable?. In this case the UAS flight plan has to be changed taken in account the new weather conditions.

In the Figure 7.1 and 7.2 we can see the different weather contingencies that we can find and how the HM and the CIC solves these problems.

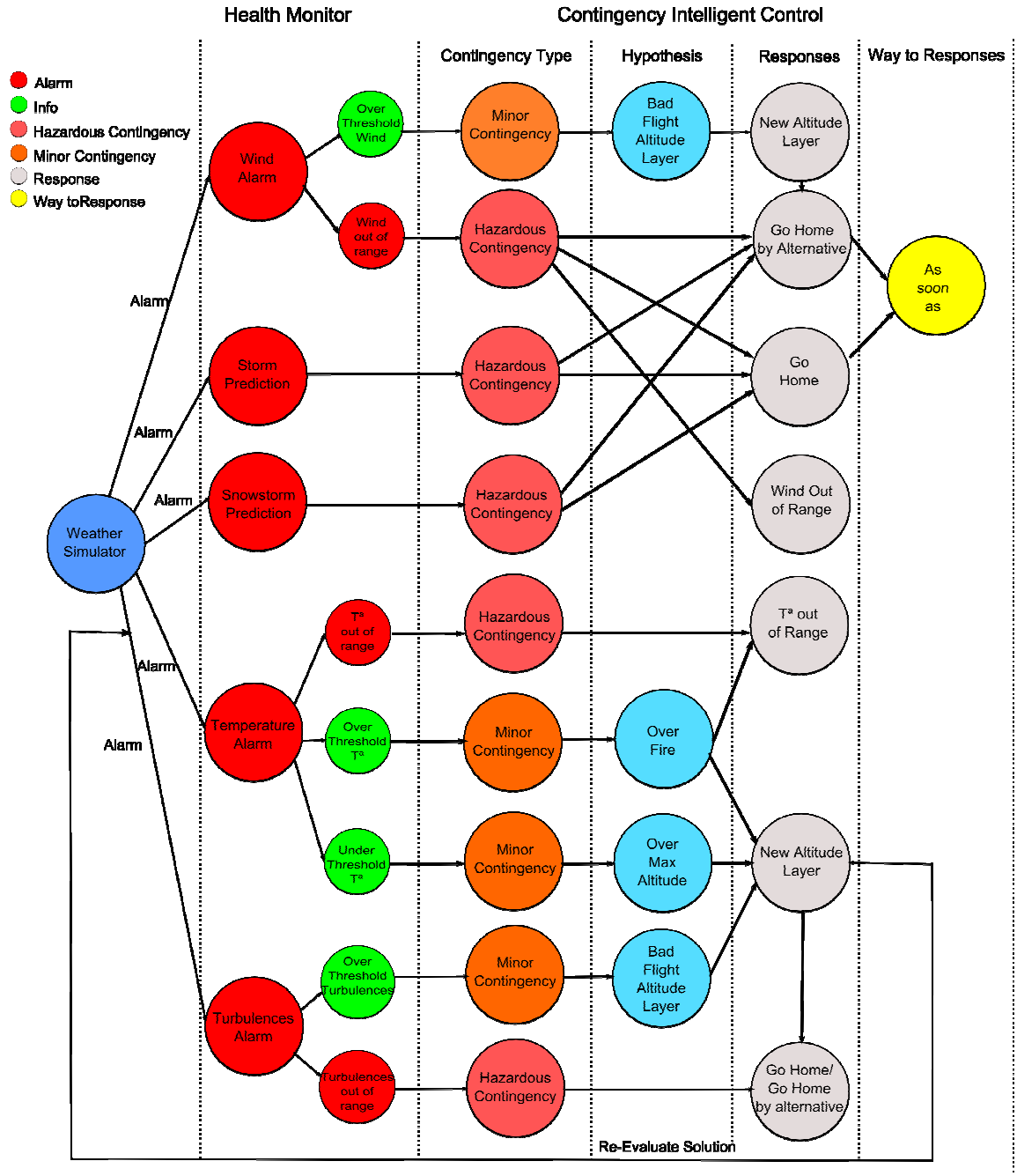

Fig.7.1 Weather use case I

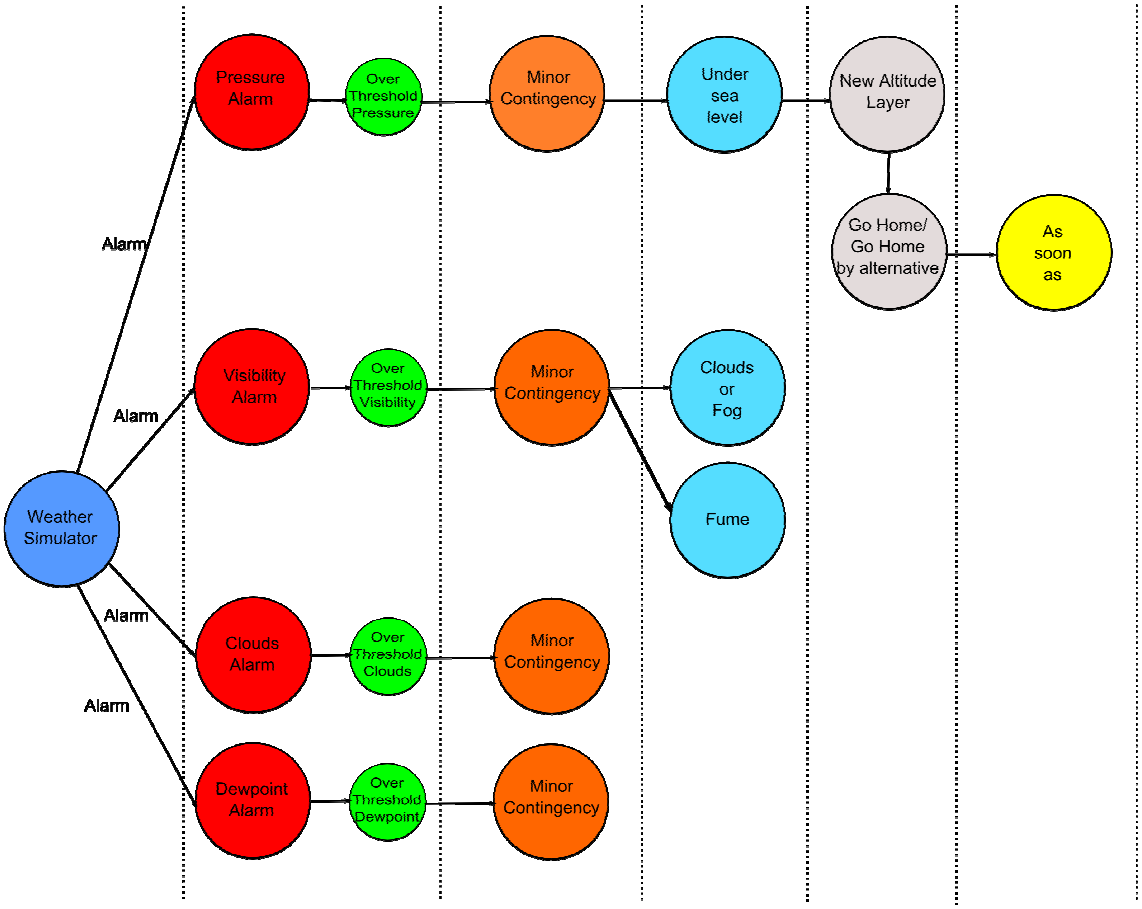

Fig.7.2 Weather use case II

Figure 7.1 and 7.2 shows the different contingencies that we can detect with the Weather Manager. In this example the Weather Manager service publishes different alarms which are detected by itself. On the other hand, through its variables as the temperature or the wind, the CM might find future contingency situations. When the temperature outside the aircraft is over a threshold, the HM raises a minor contingency. This concept is very similar for the wind or when the temperature falls down. To solve this problem the CIC will propose different responses for each minor contingency. These responses are based in a little flight plan change. For example, we might find a lot of wind in a flight plan altitude layer. Changing the flight plan altitude we can solve this contingency. If we have changed the altitude several times and the wind is over the threshold, the CIC will propose go back home. All these decisions will be supervised by the pilot in command on the USAL flight monitor.

Engine Use Cases

The UAS engine is managed by the USAL engine manager. This service controls all the engine parameters involved with the engine, such as temperature, oil, fuel, etc. The service manages the correct range of each parameter; these ranges can be configurated for each engine. When the engine manager detects any parameter out of range, an event is raised to inform the CM.
Next figures show the engine use case. It described the CM responses in front of engine paramount mishaps.

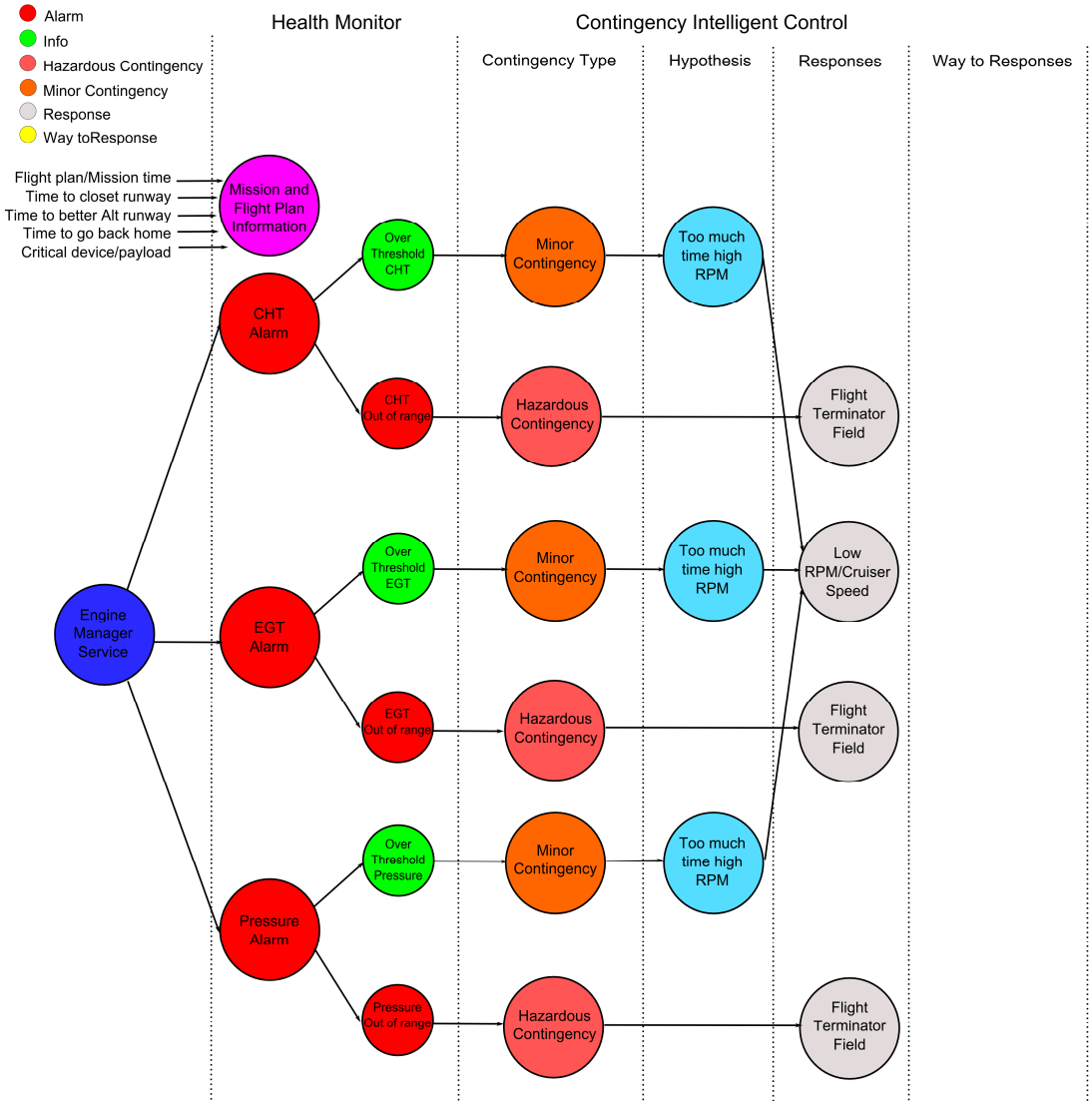

Fig.7.3 Engine use case I

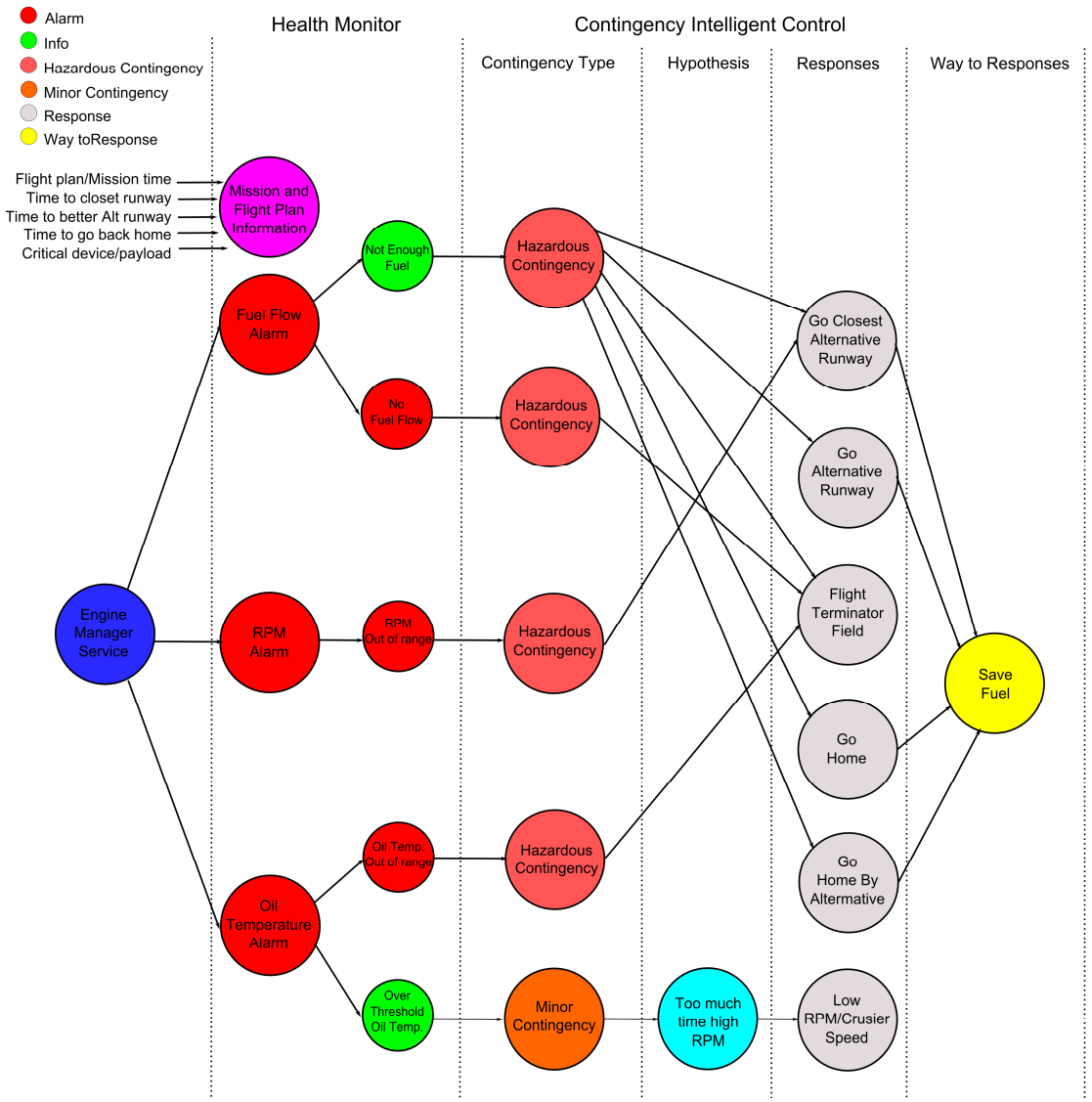

Fig.7.4 Engine use case II

Figure 7.3 and 7.4 shows several engine contingencies. In order to manage some engine contingencies it is needed mission or flight plan information. For example in the “No enough fuel” contingency; we need mission information to provide an intelligent response. If we are near home, maybe it is suitable to land over there. However, if we do not have enough fuel time to arrive home maybe the correct response is looked for the closest runway to land. The other contingencies are addressed to the engine parameters. For example when the CHT or EGT are out of range the probabilities of the engine crash are so high. Therefore, we must prepare the UAS to this crash. The same case is for the “No fuel flow” or “Main fold pressure out of range” of “RPM”.

Electrical Use Cases

In the USAL, the UAS energy is managed by the Electrical Manager Service (ELM). It is an on-board system in charge of offering a flexible power supply architecture that supports minimal reconfiguration overhead for a wide variety of UAS missions. The ELM is designed to offer a continuous monitoring of the state of the power network, and a coherent and controlled response in front of power supply contingencies.

The ELM will monitor the batteries and generator status, the power consumption of the avionics and other systems, manage the connection/disconnections of systems, and provide power availability estimations. However, what happen when the power forecast does not satisfy mission minimums power. In this case the UAS integrity is in danger. The CM always must to preserve the UAS safety and reliability. Therefore, it will monitor the time power forecast and the flight plan/mission time to ensure achievement the mission goals.

One most common contingency is the lack of electrical power. This lack of power can have different responses. These responses depend on the electrical time available in the batteries and the mission phase. If we have time to go back home, we would have to land there. On the other hand, maybe we only have enough time to go the closest runway or search a flight termination field in the worst case. Therefore, the CM needs flight plan/mission information. This paramount information will be very useful in order to take any decision. Next figures show the electrical use case.

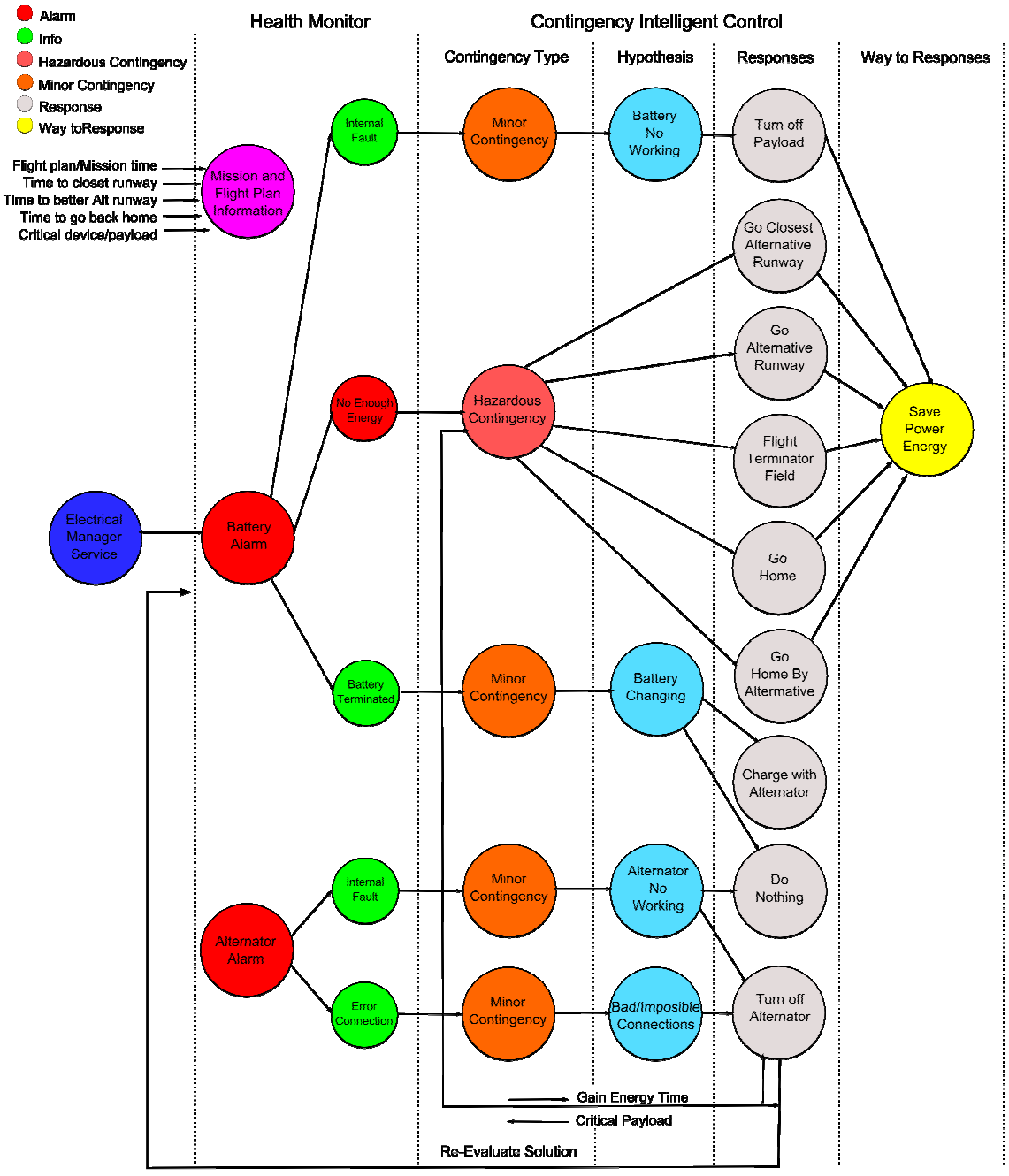

Fig.7.5 Electrical use case I

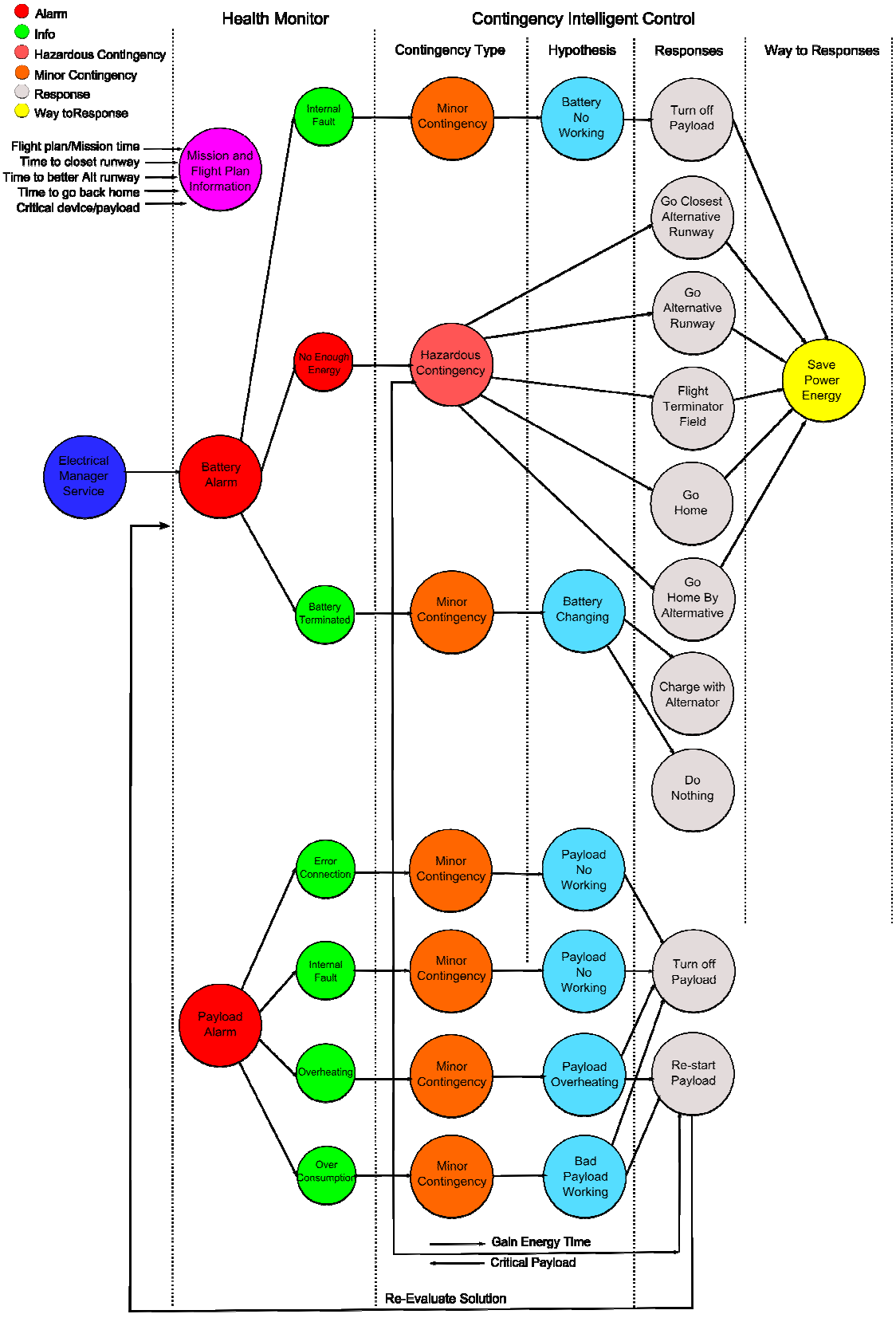

Fig.7.6 Electrical use case II

As it is shown in the figure 7.5 and 7.6 with the “energy time” and flight plan information the CM should provide an intelligent response in front of no enough energy contingency. The CM will know if the aircraft might arrive home, or if it has to land in the closest runway. When the battery is terminated, the ELM will charge it. In this way the CM knows the batteries status and if something is wrong the CM can command the batter charging. On the other hand, the ELM manages different electrical alarms related with devices. Alarms as “No Payload Consumption” or “payload over/under consumption” show failure in any device. Depends on how important is each device, this type of contingency needs different responses. For example, if we need this payload for the mission, then the mission has to be canceled. However, if the payload is not critical for the mission, it might be turned off. Finally, when the UAS has a lack of energy contingency, it has to be treated in the “save power energy” way. In this mode the UAS treats to provide energy the critical devices, as for example the autopilot, turning off the rest of devices.

SECTION 8. VARIABLES AND EVENTS PUBLISHED

This section is going to explain the all variables and events that are published in the middleware. The first three subsections explain the variables and events that have published each of the simulators. They also show that service is subscribed to each of the variables published. The last subsection explains the events and the variables published by the Contingency System.

8.1 Variables and Events of the Weather Simulator

This section explains the all variables and events that are published by the Weather Simulator. The variables are represented in the Table 8.1 and the events are represented in the Table 8.2. As shown in the both tables, the right column shows the services that have subscribed to variables or events of the Weather Simulator.

	Variables	**Subscribed Service**
Weather Simulator	Wind	Contingency System
	Elevation	Contingency System
	Turbulences	Contingency System
	Visibility	Contingency System
	Rain	Contingency System
	Snow	Contingency System
	Clouds	Contingency System
	Dew point	Contingency System
	Temperature	Contingency System
	Pressure	Contingency System

Table 8.1 Variables published by the Weather Simulator.

	Events	Subscribed Service
Weather Simulator	Alarm_Wind	Contingency System
	Alarm_Dewpoint	Contingency System
	Alarm_Turbulences	Contingency System
	Alarm_Visibility	Contingency System
	Alarm_Rain	Contingency System
	Alarm_Snow	Contingency System
	Alarm_Cloudy	Contingency System
	Alarm_Temperature	Contingency System
	Alarm_Pressure	Contingency System

Table 8.2 Events published by the Weather Simulator.

In the two tables above, we can see that the Contingency System is the only service subscribed to the variables and events of the Weather Simulator. The Contingency System needs these values to classify the alarms and determine a solution to the UAS.

8.2 Variables and Events of the Engine Simulator

This section explains the all variables and events that are published by the Engine Simulator. The variables are represented in the Table 8.3 and the events are represented in the Table 8.4. As shown in the both tables, the right column shows the services that have subscribed to variables or events of the Engine Simulator.

	Variables	**Subscribed Service**
Engine Simulator	RPM	Contingency System
	CHT	Contingency System
	EGT	Contingency System
	Oil Pressure	Contingency System
	Oil_Temperature	Contingency System
	Fuel_Flow	Contingency System

Table 8.3 Variables published by the Engine Simulator.

	Events	**Subscribed Service**
Engine Simulator	Alarm_RPM	Contingency System
	Alarm_CHT	Contingency System
	Alarm_OilTemperature	Contingency System
	Alarm_OilPressure	Contingency System
	Alarm_FuelFlow	Contingency System
	Alarm_Fuel	Contingency System

Table 8.4 Events published by the Engine Simulator.

In this subsection occurs as in the previous subsection, the Contingency System is the only service subscribed to the variables and events of the Engine Simulator.

8.3 Variables and Events of the Electrical Simulator

This section explains the all variables and events that are published by the Electrical Simulator. The variables are represented in the Table 8.5 and the events are represented in the Table 8.6. As shown in the both tables, the right column shows the services that have subscribed to variables or events of the Electrical Simulator.

	Variables	**Subscribed Service**
Electrical Simulator	Alternator	Contingency System
	LeadBattery	Contingency System
	GPS	Contingency System
	PhotoCamera	Contingency System
	VideoCamera	Contingency System
	VideoTrans	Contingency System
	ThermalCamera	Contingency System
	LipoBattery	Contingency System
	Autopilot	Contingency System
	Servos	Contingency System

Table 8.5 Variables published by the Electrical Simulator.

	Events	**Subscribed Service**
Electrical Simulator	Alarm_Photo	Contingency System
	Alarm_Video	Contingency System
	Alarm_Thermal	Contingency System
	Alarm_Trans	Contingency System
	Alarm_Gps	Contingency System
	Alarm_Auto	Contingency System
	Alarm_Servos	Contingency System

	Alarm_LeadBattery	Contingency System
	Alarm_LipoBattery	Contingency System
	Alarm_Alternator	Contingency System

Table 8.6 Events published by the Electrical Simulator.

In this subsection occurs as in the previous subsection, the Contingency System is the only service subscribed to the variables and events of the Electrical Simulator.

8.4 Variables and Events of the Contingency System

This section explains the all variables and events that are published by the Contingency System. The variables are represented in the Table 8.7 and the events are represented in the Table 8.8. As shown in the both tables, the right column shows the services that have subscribed to variables or events of the Contingency System.

	Variables	**Subscribed Service**
Contingency System	Alternator_Contingency	Electrical Simulator
	LeadBattery_Contingency	Electrical Simulator
	GPS_Contingency	Electrical Simulator
	PhotoCamera_Contingency	Electrical Simulator
	VideoCamera_Contingency	Electrical Simulator
	VideoTrans_Contingency	Electrical Simulator
	ThermalCamera_Contingency	Electrical Simulator
	LipoBattery_Contingency	Electrical Simulator
	Autopilot_Contingency	Electrical Simulator
	Servos_Contingency	Electrical Simulator
	New_altitude_layer	Flight Plan Manager Simulated
	reset_boton_fuel	Engine Simulator

Table 8.7 Variables published by the Contingency System.

	Events	Subscribed Service
Contingency System	Alarm_GoHome	Flight Plan Manager Simulated
	Alarm_FlightTerminatorField	Flight Plan Manager Simulated
	Alarm_GoClosetAlternativeRunway	Flight Plan Manager Simulated
	Alarm_GoHomeByAlternative	Flight Plan Manager Simulated
	Alarm_GoAlternativeTunway	Flight Plan Manager Simulated

Table 8.8 Events published by the Contingency System.

The variables published by the Contingency System are used to turn off the alarms of the simulators. They will know that the problem is resolved through the variables that publish the Contingency System.

The events published by the Contingency System are used to determine the action that the UAS has to take. The Contingency System decides to publish a particular event when a particular hazardous alarm is produced.

8.5 Variables of the Flight Plan Manager Simulated

This section explains the new variables that are published by the Flight Plan Manager Simulated. We have not create this service, FPMS is already existed. We needed to expand the FPMS code to make the integration of our services in the ISIS. This has led to publish new variables.

The variables are represented in the Table 8.9. As shown in the Table 8.9, the right column shows the services that have subscribed to variables of the FPMS.

	Variables	**Subscribed Service**
FPMS	Reset_altitude_layer	Contingency System
	Times	Contingency System
	Reset_Alarm_FlightTerminatorField	Contingency System
	Reset_Alarm_GoAlternativeRunway	Contingency System
	Reset_Alarm_GoClosetAlternativeRunway	Contingency System
	Reset_Alarm_GoHome	Contingency System
	Reset_Alarm_GoHomeByAlternative	Contingency System

Table 8.9 Variables published by the FPMS.

In this subsection occurs as in the Electrical Simulator subsection, the Contingency System is the only service subscribed to the variables of the FPMS.